Tensor Proxies for Efficient Feature Cross Search

Anonymous Author(s)
Affiliation
Address
email

Abstract

Feature crossing is a popular method for augmenting the feature set of a machine learning model by taking the Cartesian product of a small number of existing categorical features. While feature crosses have traditionally been hand-picked by domain experts, a recent line of work has focused on the automatic discovery of informative feature crosses. Our work proposes a simple yet efficient and effective approach to this problem using *tensor proxies* as well as a novel application of the *attention mechanism* to convert the combinatorial problem of feature cross search to a continuous optimization problem. By solving the continuous optimization problem and then rounding the solution to a feature cross, we give a highly efficient algorithm for feature cross search that trains only a single model for feature cross searching, unlike prior greedy methods that require training a large number of models. Through extensive empirical evaluations, we show that our algorithm is not only efficient, but also discovers more informative feature crosses that allow us to achieve state-of-the-art empirical results for feature cross models. Furthermore, even without the rounding step, we obtain a novel DNN architecture for augmenting existing models with a small number of features to improve quality *without introducing any feature crosses*. This avoids the cost of storing additional large embedding tables for these feature crosses.

1 Introduction

The idea of introducing nonlinear augmentations of feature sets to improve model quality is a widely used technique in machine learning, with various powerful instantiations of this idea ranging from the classic kernel trick to more complex methods for image augmentation. For categorical features, one version of this technique is *feature crossing*, in which one introduces the Cartesian product of existing categorical features as a new feature. For instance, if a training example consists of two features x and y, then the pair (x, y) is formed as a new feature. It is well-known that this additional nonlinearity is a powerful method for improving the predictive capacity of machine learning models.

Traditionally, feature crosses are constructed manually by domain experts and require extensive human intervention. While this is a successful approach in many cases, it is far more desirable to construct informative feature crosses algorithmically. Thus, a recent line of work has focused on developing efficient algorithms for feature cross search [LWZ+19, LLZ20, CEF+21].

A key challenge for feature cross search algorithms is in the combinatorial nature of the problem, in which one must search over a space of size $\binom{m}{k}$ in order to find an optimal cross of k features among a pool of m base features. Thus, the search space is exponential in k, and in fact, certain versions of the feature cross problem have been shown to be NP-hard even to approximate to a superconstant factor [CEF+21]. Thus, heuristics and greedy algorithms are usually considered in order to obtain tractable algorithms [LWZ+19, LLZ20, CEF+21]. However, even such greedy approaches require one to form many feature crosses and train many models, which makes application to large-scale settings difficult.

Submitted to ICML 2023 Workshop: Sampling and Optimization in Discrete Space. Do not distribute.

1.1 Our contributions

In this work, we propose a novel algorithm for efficient feature cross search through the use of *tensor proxies*. At a high level, this approach introduces a continuous proxy for feature crosses that can be optimized via standard gradient-based optimization techniques, and rounds this proxy to a feature cross at the end of training this model. The tensor proxies that we introduce are solely a function of the feature embeddings of the original base features, and are therefore much more computationally efficient compared to their feature cross counterparts, which typically require introducing large trainable embedding tables for each candidate feature cross. Furthermore, in many cases, we show that our method in fact provides model quality improvements *without rounding the tensor proxies*, which leads to further efficiency improvements. We highlight that our techniques for augmenting models with feature crosses and tensor proxies are especially useful in settings where we have a base deep neural network (DNN) under consideration that we would like to improve, but we do not want to change the model substantially; this could be, for example, if the base DNN has already been heavily optimized in various aspects, or is tied to specific hardware. In such settings, introducing a small number of feature crosses or tensor proxies allows for lightweight changes to the model that can substantially improve prediction quality.

Our algorithm builds on the standard method of using *feature embeddings* to map categorical features to numerical vector-valued features (see, e.g., [WDL+09, NMS+19]). That is, for a categorical feature with a vocabulary size of q and an embedding dimension d, we consider an *embedding matrix* $\mathbf{E} \in \mathbb{R}^{q \times d}$ and map the categorical feature value $j \in [q]$ to the vector $\mathbf{e}_j^\top \mathbf{E} \in \mathbb{R}^d$ given by the j-th row of $\mathbf{E}$. This allows us to replace the categorical feature by a d-dimensional vector-valued feature, which allows for the application of a standard DNN model on top of these embedded features. The embedding matrix $\mathbf{E}$ is often trained together with the DNN that consumes these embedded features.

1.1.1 Tensor proxy features

With feature embeddings of categorical features in hand, we can now introduce the idea of tensor proxies. Suppose that we have a set of k features to cross. Then, instead of forming the Cartesian product of the k features and then embedding this cross feature, we consider the following *proxy* feature cross that is formed only as a function of the embeddings of the original base features.

Definition 1.1 (Tensor proxy features)**.** Let $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)} \in \mathbb{R}^d$ be a set of k vector-valued features in d dimensions. Let $\mathcal{B} \in \mathbb{R}^{d \times d \times \cdots \times d}$ be an order-k *core tensor* that has dimension d in each of its k modes. Then, we define the *tensor proxy feature* associated with the core tensor $\mathcal{B}$ to be

$$\mathsf{TP}_{\mathcal{B}}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}) \coloneqq \mathcal{B} \times_1 \mathbf{x}^{(1)} \times_2 \mathbf{x}^{(2)} \cdots \times_k \mathbf{x}^{(k)}$$

where $\mathcal{B} \times_i \mathbf{x}^{(i)}$ denotes the tensor-vector multiplication of $\mathcal{B}$ and the vector $\mathbf{x}^{(i)}$ along mode i.

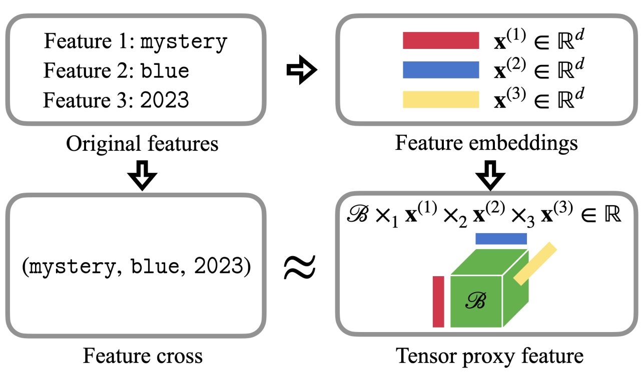

Figure 1: Tensor proxy features for efficiently approximating higher-order feature interactions.

In our tensor proxy features of Definition 1.1, we take the core tensor $\mathcal{B}$ to be simultaneously trained together with the DNN model consuming these tensor proxy features. Thus, our objective is to learn a combinatorial feature which captures k-th order interactions between k features using just the feature embeddings of the original base features, without forming the feature cross of the k features. This tensor proxy feature is intended to serve as a cheap proxy to capture the information captured by the feature cross of these k features.

Note that introducing the core tensor $\mathcal{B}$ is often far more efficient than introducing a new embedding table $\mathbf{E}$ for the feature cross. Indeed, a typical setting might involve the crossing of three features, each with a vocabulary size of, say, $q = 1000$, which are each embedded into a dimension of $d = 10$. Then, the embedding table $\mathbf{E}$ for the feature cross of these features could be as large as $d \cdot q^3 = 10^{10}$.[1] In contrast, the core tensor $\mathcal{B}$ would only involve introducing an additional $d^3 = 10^3$ parameters.

Remark 1.2. In Definition 1.1, we have introduced a tensor proxy model that is inspired by a *Tucker tensor decomposition*. However, one can just as easily define a number of other variants of our tensor proxy features based on other common tensor decompositions such as the CP decomposition and general tensor networks [MWZ22]. We leave the exploration of variations on the parameterization of the tensor in tensor proxy features as an exciting direction for future work.

While tensor proxies are just one way of defining a model architecture for representing a surrogate feature cross, our empirical evaluations suggest that by modeling the Cartesian product in feature crosses by a tensor product, we obtain an especially effective model architecture for this task. Indeed, we compare our method with a similar method which uses a small DNN to serve as a proxy to feature crosses, and we show that tensor proxies provide superior empirical performance.

1.1.2 Learning tensor proxy features

While tensor proxies (Definition 1.1) allow us to avoid forming and embedding feature crosses, we are still left with a combinatorial optimization problem where we must optimize a set function over subsets of $[m]$ of size k. In Section 2, we introduce our various approaches for this combinatorial optimization problem of learning tensor proxies. In particular, we discuss the following algorithms:

- **Greedy search**: Much of the prior work on feature cross search focused on greedy algorithms [LWZ+19, LLZ20, CEF+21] based on learning an order-k feature cross by iteratively learning order j feature crosses for $j \in [k]$. In this algorithm, given a feature cross $S \subseteq [m]$ of order-j, one constructs a feature cross of order $j + 1$ by evaluating m feature cross candidates $S \cup \{i\}$ for $i \in [m]$ and selects the best candidate. This is closely related to a heuristic known as *beam search*. While these algorithms perform quite well in practice, they require training mk models, which can be prohibitively expensive in large-scale settings.
- **Sequential Attention for tensor proxy search**: In order to speed up greedy search algorithms for feature selection, the recent work of [YBC+23] introduced the *Sequential Attention* algorithm, which uses a variation on the attention mechanism to efficiently simulate the greedy algorithm. In particular, this work shows that the greedy process of considering m individual feature candidates and then selecting the best candidate can be simulated by training a single model that multiplies each of the m feature candidates by a trainable softmax mask (or "attention weights"), and then selecting the feature with the largest attention weight. We consider an adaptation of this algorithm to the setting of learning tensor proxies. Note that this idea reduces the number of model trainings to just k, which is substantially more efficient than a naïve greedy search algorithm.
- **Simultaneous tensor proxy search**: Finally, inspired by the use of the attention mechanism in the Sequential Attention algorithm [YBC+23], we consider a novel attention-based search algorithm which takes advantage of the tensor structure of the tensor proxy features in order to train tensor proxy features in a *single model training*. In this algorithm, we construct a single tensor proxy feature that uses an attention-inspired architecture to simultaneously search over the space of all k vector-valued features, bypassing the greedy search process used in the previous two algorithms. Our algorithm has the potential to discover powerful new feature crosses that are "hidden" to greedy algorithms, whose informative value only appears when all k features of the feature cross are crossed together, but not when only a subset of the k features are crossed. Further, the efficiency improvements of this algorithm are even more marked when we wish to discover multiple feature crosses, and only requires a single round of model training to discover t feature crosses, whereas the prior two greedy

[1] Large vocabulary sizes are often dealt with by hashing the vocabulary into a smaller set [WDL+09]. Trade-offs concerning the hash table size are outside the scope of this work, and our discussion will ignore this aspect for simplicity, although our experiments do use hashing.

methods require training t times as many models; that is, the prior two algorithms require training mkt and kt models, respectively, while the simultaneous tensor proxy search only requires 1 model training (see Table 3).

1.1.3 Rounding tensor proxy features

As a final component of our tensor proxy framework for feature cross search, we show that by mapping the tensor proxy features discovered by the algorithms discussed previously to their corresponding actual feature crosses and retraining the resulting model, we obtain state-of-the-art feature cross-augmented models. Thus, this demonstrates that our proposed tensor proxy features of Definition 1.1 provide high quality proxies that allow us to replace the expensive operation of constructing feature crosses and the associated combinatorial optimization problem with computationally inexpensive proxies, which can be efficiently optimized using continuous optimization techniques.

In fact, we show that even without the rounding step, tensor proxy features provide a novel model architecture that can be used to augment existing models in a computationally inexpensive way. This augmentation can significantly improve model quality, and in some cases even outperform feature cross models. This is because tensor proxies are inexpensive to optimize over and use in the model, as they do not require the large embedding matrices that are typically used for feature crossing.

1.2 Novelty and comparisons to related work

The problem of designing efficient model architectures for prediction on categorical data is a ubiquitous problem with important applications including recommendation systems, natural language processing (NLP), and click-through-rate (CTR) prediction, and has been studied intensely in many works. In particular, our work is partially inspired by a long line of work initiated by [Ren10], which observes that DNN architectures that form polynomials of feature embeddings provide a powerful way to efficiently improve model quality [WFFW17, XYH+17, GTY+17, LZZ+18, NMS+19, SSX+19, CSH20, WSC+21, CWL+21]. This problem has also been referred to as the problem of learning *feature interactions* or *combinatorial features* [SSX+19].

Prior work on forming higher-order combinatorial features typically takes the approach of designing some feature combination layer that represents a degree-2 polynomial, and then composing these layers by stacking them on top of each other like a DNN. However, this type of model architecture precludes their use in our application for efficient feature cross search, since this makes it difficult to isolate the contribution of a fixed order-k feature cross (or some proxy of the feature cross). Thus, one point of novelty in our work lies in our new parameterization of these combinatorial features via our tensor proxy feature definition, which exploits a novel tensor-based structure and allows us to directly use these polynomial features as proxies for feature crosses.

Another important line of work that is closely related to our work is the idea of using the *attention mechanism* for DNN architectures [VSP+17]. In the context of learning feature interactions, [SSX+19] explored the idea of using self-attention to form linear combinations of features that are the most relevant to each feature. In [YBC+23], a substantially simplified version of the attention mechanism is used for a feature selection algorithm.

While our attention-based algorithms for tensor proxy search are inspired by the work of [YBC+23], we depart from their approach in multiple important ways, as we discuss further in Section 2. Most notably, our simultaneous tensor proxy search algorithm uses the structure of the tensor proxy feature to simultaneously optimize over (a relaxation of) the space of all $\binom{m}{k}$ tensor proxy features, and avoids the greedy process that is prone to missing features that provide informative value only when all k features are crossed together, but not informative on any smaller subset of features.

Finally, tensor proxies (Definition 1.1) are inspired by a long line of work on the study of using tensor decompositions for efficient machine learning [KLK+20, SBK+20, KKP+21, MSM+21, FFG22, GFFM23]. While prior work has focused on the direct application of tensor decompositions to compress and denoise a dataset or weight tensor, our central contribution in this work is to provide a novel connection between tensor decompositions and feature crosses, which allows us to exploit the continuous and algebraic structure of tensors to efficiently solve the combinatorial problem of feature cross search.

2 Algorithms for learning tensor proxy features

In this section, we discuss our proposed algorithms for learning tensor proxy features. For the analogous problem of feature cross search, natural greedy search algorithms have been considered in many prior works [LWZ+19, CEF+21]. This immediately translates to an algorithm in the setting of tensor proxy features. However, in this work, we seek algorithms which are much more efficient, by exploiting the algebraic structure of tensor proxy features.

2.1 Sequential Attention for tensor proxy search

We first consider a more efficient method of implementing the greedy search algorithm, based on the *Sequential Attention* algorithm in [YBC+23]. In this algorithm, suppose that we have already selected a subset $S \subseteq [m]$ of j features to be included in the final tensor proxy feature. Then, we select the $(j+1)$-th feature as follows. We first form m candidate tensor proxies given by

$$\mathsf{TP}_{\mathcal{B}^i}(\{\mathbf{x}^{(\ell)}\}_{\ell \in S \cup \{i\}}), \qquad i \in [m].$$

At this point, the algorithm is still the same as the classical greedy search algorithm, and the greedy algorithm would proceed by evaluating the m models (each of which consumes one tensor proxy candidate) and selecting the best one. However, instead, the Sequential Attention algorithm trains a single model that consumes m tensor proxy features given by

$$\mathrm{softmax}(\mathbf{w})_i \cdot \mathsf{TP}_{\mathcal{B}^i}(\{\mathbf{x}^{(\ell)}\}_{\ell \in S \cup \{i\}}), \qquad i \in [m], \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^m$ are trainable weights. Intuitively, this allows the model to simultaneously consider all m tensor proxy candidates, and continuously add weight to the most informative tensor proxy. At the end of training, we select the tensor proxy with the largest attention weight to include in our model.

Algorithm 1 Sequential Attention for tensor proxy search.

1: **function** SEQATTTP(dataset $\mathbf{X} \in \mathbb{R}^{n \times m \times d}$, labels $\mathbf{y} \in \mathbb{R}^n$, tensor proxy order k)
2: $\quad S \leftarrow \varnothing$ $\qquad \triangleright$ Selected features
3: $\quad$ **for** $\ell \in [k]$ **do**
4: $\qquad$ Let $\mathbf{w} \in \mathbb{R}^m$ and $\mathcal{B}^i \in \mathbb{R}^{d \times d \times \cdots \times d}$ for $i \in [m]$ be trainable weights
5: $\qquad$ Train a model using the features $\mathbf{X}$ and $\mathrm{softmax}(\mathbf{w})_i \cdot \mathsf{TP}_{\mathcal{B}^i}(\{\mathbf{x}^{(\ell)}\}_{\ell \in S \cup \{i\}})$ for $i \in [m]$
6: $\qquad S \leftarrow S \cup \{\mathrm{argmax}(\mathrm{softmax}(\mathbf{w}))\}$
7: $\quad$ **return** S

While this algorithm is more efficient than classic greedy algorithm, note that it still does not exploit the structure of tensor proxy features, and in fact immediately implies a corresponding algorithm for the standard feature cross problem as well, just by replacing each tensor proxy with the corresponding feature cross. Another shortcoming of this approach is that, naïvely, the Sequential Attention algorithm for tensor proxy search requires training k models. As suggested by [YBC+23], it is possible to run this algorithm in one model training just by using a $1/k$ fraction of the training set to select each of the k features. However, this decreases the amount of training data used to train each feature, and could be disadvantageous in some settings. Furthermore, the greedy structure of the Sequential Attention algorithm poses the possible problem that if an important feature cross of order k has the property that any subset of $k-1$ of its features does not provide an informative feature cross, then it is unlikely to be discovered (see discussion of this phenomenon in, e.g., [CEF+21]). Another problem occurs when we want to add t feature crosses to the model instead of just one, in which case we must repeat this entire algorithm t times.

2.2 Simultaneous tensor proxy search

To address the shortcomings of a naïve greedy search for tensor proxies as well as the Sequential Attention-based optimized greedy algorithm, we consider a novel attention-based algorithm to *simultaneously* search over the space of order-k tensor proxy features.

The key idea lies in using the attention weights in a different way than in Equation (1). Instead of using the attention weights at the level of the tensor proxies, our crucial insight is to use the attention weights *at the level of the feature embeddings*. That is, we introduce k sets of weights $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(k)} \in \mathbb{R}^m$, and consider k *mixed attention-weighted* feature embeddings given by

$$\sum_{j=1}^{m} \mathrm{softmax}(\mathbf{w}^{(\ell)})_j \cdot \mathbf{x}^{(j)}, \qquad \ell \in [k].$$

Then, we train a single model that consumes a single tensor proxy feature given by

$$\mathsf{TP}_{\mathcal{B}}\left(\left\{\sum_{j=1}^{m} \mathrm{softmax}(\mathbf{w}^{(\ell)})_j \cdot \mathbf{x}^{(j)}\right\}_{\ell=1}^{k}\right), \tag{2}$$

where each of the k feature crosses used by the tensor proxy feature is an independent attention-weighted feature embedding. Intuitively, the attention weights will give more weight towards the features that provide the most predictive value when used as a component of the tensor proxy. Finally, at the end of training, we round the attention weights to a selection of k tensor proxy features which only take k pure feature embeddings as input.

Algorithm 2 Simultaneous tensor proxy search.

1: **function** SIMULTANEOUSTP(dataset $\mathbf{X} \in \mathbb{R}^{n \times m \times d}$, labels $\mathbf{y} \in \mathbb{R}^n$, tensor proxy order k)
2: Let $\mathbf{w}^{\ell} \in \mathbb{R}^m$ for $\ell \in [k]$ and $\mathcal{B} \in \mathbb{R}^{d \times d \times \cdots \times d}$ be trainable weights
3: Let $\mathbf{X}^{(j)} = \mathbf{X}[:, j, :] \in \mathbb{R}^{n \times d}$ be the j-th feature for $j \in [m]$
4: Let $\mathbf{Z}^{(\ell)} \leftarrow \sum_{j=1}^{m} \mathrm{softmax}(\mathbf{w}^{\ell})_j \cdot \mathbf{X}^{(j)}$ for $\ell \in [k]$
5: Train a model using the the features $\mathbf{X}$ and $\mathsf{TP}_{\mathcal{B}}(\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(k)})$
6: $S \leftarrow \{\mathrm{argmax}(\mathrm{softmax}(\mathbf{w}^{\ell})) : \ell \in [k]\}$ ▷ Selected features
7: **return** S

Finally, note that unlike the greedy and Sequential Attention algorithms, extending Algorithm 2 to discover t feature crosses rather than one is trivial—we just train our base model with t independent units $\mathsf{TP}_{\mathcal{B}}(\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(k)})$ added to its feature set (see Line 5), rather than just one.

2.2.1 Connections to learning monomials

We note that our simultaneous tensor proxy search algorithm can be viewed as a generalization of a natural algorithm for learning monomials. Indeed, we can consider the natural problem of learning an order-k monomial f over m variables given by

$$f(\mathbf{x}) = \prod_{\ell \in S} \mathbf{x}_\ell,$$

for some subset $S \subseteq [m]$ of size k, when we are given a dataset $\mathbf{X} \in \mathbb{R}^{n \times m}$ as well as observations of the monomial f as evaluations $\mathbf{y}_i = f(\mathbf{e}_i^\top \mathbf{X})$ for each example $i \in [n]$. This is a specific instance of the problem of learning monomials [ADHV19] and sparse polynomials [APVZ14a, APVZ14b] from their evaluations, which has received much attention in the theory community.

One possible algorithm for learning the monomial f is to optimize the (nonconvex) function

$$\min_{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(k)} \in \mathbb{R}^m} \sum_{i=1}^{n} \mathcal{L}\left(\prod_{\ell=1}^{k} \left\langle \mathbf{e}_i^\top \mathbf{X}, \mathbf{w}^{(\ell)} \right\rangle, \mathbf{y}_i\right), \tag{3}$$

where $\mathcal{L}(\cdot, \cdot)$ is some loss function. Note that in the setting where we have noiseless observations $\mathbf{y}_i = f(\mathbf{e}_i^\top \mathbf{X})$ and $\mathcal{L}$ is a nonnegative loss function that vanishes only when its two arguments are equal, such as the ℓ_2 distance, then setting the $\mathbf{w}^{(\ell)}$ for $\ell \in [k]$ to $\mathbf{e}_\ell$ for $\ell \in S$ provides a global minimizer of this objective. Thus, solving the optimization problem given by (3) and then rounding the variables $\mathbf{w}^{(\ell)}$ to standard basis vectors is a natural approach to learning monomials.

In fact, it is not hard to see that this monomial learning algorithm exactly corresponds to our tensor proxy framework when all embedding dimensions are $d = 1$, and the softmax masks $\mathrm{softmax}(\mathbf{w}^{(\ell)})$ in (2) are replaced by their logits $\mathbf{w}^{(\ell)}$. One interesting question is whether this algorithm can indeed recover monomials or not, when the observations $\mathbf{X} \in \mathbb{R}^{n \times m}$ are given by i.i.d. Gaussian random variables. While we do not resolve this question, we believe it is an interesting direction of future research. In fact, this problem may be interesting even for the order $k = 2$ case and the least squares loss. Note that this case is a special case of rank-1 matrix sensing, for which recent work on nonconvex optimization has succeeded in showing that for design matrices with the restricted isometry property, SGD on the rank-1 factors can successfully recover the desired optimal rank 1 matrix (see, e.g., [BNS16, GJZ17]). However, showing analogous results for the monomial learning problem may be difficult due to the lack of a restricted isometry property for the Khatri–Rao power of a Gaussian matrix, even though a Gaussian matrix itself does satisfy the restricted isometry property.

3 Experiments

In this section, we provide extensive empirical evaluations of our tensor proxy-based feature cross search algorithms. Our experiments are conducted on 7 popular public datasets for categorical classification tasks that have been considered in many prior works. The size and sources of our datasets and other details concerning the experiments are provided in Appendix A.

In Tables 1 and 2, we first evaluate the performance of 9 baseline algorithms including those introduced in the works of [WFFW17, WSC+21, CWL+21, NMS+19, CSH20, SSX+19, GTY+17, LZZ+18]. These baseline algorithms are some of the most popular and successful model architectures for prediction on categorical datasets. We note that these baselines simply propose a DNN model architecture for a given set of features, and thus are independent of feature crosses.

Table 1: Baseline AUC performance of benchmark algorithms.

	Adult	Bank	Credit	Employee	Frappe	Avazu	Criteo
MLP	0.8985	0.9298	0.8624	0.7590	0.9816	0.7315	0.7926
DCN [WFFW17]	0.8845	0.9298	0.8620	0.7239	**0.9819**	0.7314	0.7915
DCNv2 [WSC+21]	0.8976	0.9260	0.8611	0.7808	0.9802	**0.7326**	0.7923
EDCN [CWL+21]	0.9043	0.9154	**0.8635**	0.7196	0.9790	0.7324	**0.7934**
DLRM [NMS+19]	0.8659	0.9269	0.8604	0.7163	0.9806	0.7305	0.7905
AFN [CSH20]	**0.9097**	**0.9324**	0.8633	0.7459	0.9771	0.7286	0.7892
AutoInt [SSX+19]	0.9001	0.9253	0.8604	0.7188	0.9761	0.7316	0.7914
DeepFM [GTY+17]	0.8951	0.8799	0.8600	**0.8064**	0.9806	0.7283	0.7890
xDeepFM [LZZ+18]	0.9004	0.8954	0.8588	0.7964	0.9806	0.7313	0.7903

Table 2: Baseline loss performance of benchmark algorithms.

	Adult	Bank	Credit	Employee	Frappe	Avazu	Criteo
MLP	0.3914	0.2645	0.1775	0.2252	0.1843	0.4093	**0.4684**
DCN [WFFW17]	0.4439	0.2696	0.1777	0.2199	**0.1810**	0.4186	0.4691
DCNv2 [WSC+21]	0.4171	0.2619	0.1774	0.2150	0.1858	0.4143	0.4685
EDCN [CWL+21]	0.5726	0.3716	**0.1771**	0.2612	0.1834	0.4214	0.4685
DLRM [NMS+19]	0.5392	0.3421	0.1783	0.2227	0.1931	0.4061	0.4762
AFN [CSH20]	**0.3131**	**0.2333**	0.1781	0.2862	0.1880	**0.3951**	0.4789
AutoInt [SSX+19]	0.4578	0.3310	0.1778	0.2579	0.2033	0.4176	0.4686
DeepFM [GTY+17]	0.3605	0.2724	0.1789	0.2076	0.1903	0.4051	0.4730
xDeepFM [LZZ+18]	0.3735	0.3158	0.1796	**0.2040**	0.1903	0.4196	0.4788

In Table 4 and Table 5, we evaluate the best models achieved by using tensor proxies (TP). In order to evaluate whether our TPs offer improved approximations of feature crosses over traditional neural networks, we also compare TPs to an analogous algorithm which uses a small neural network as the feature cross proxy, which we call neural network proxies (NP). In NPs, the interactions between k features is modeled by feeding the k feature embeddings through a small neural network.

In our experiments, we consider the addition of $t = 5$ feature crosses of order $k = 3$ to all of the baseline models considered in Tables 1 and 2. We also provide a comparison to the AutoCross feature crossing algorithm [LWZ+19] implemented with improved efficiency by combining with the Sequential Attention algorithm [YBC+23], as described in Algorithm 1 and Section 2. We note that almost all of the datasets that we consider consist of at least $m = 10$ features, and thus directly implementing the AutoCross algorithm would require sequentially training at least $mkt = 10 \cdot 3 \cdot 5 = 150$ models, which is too inefficient for our purposes. Using the idea in [YBC+23] (see Algorithm 1), we bring this number down to $kt = 3 \cdot 5 = 15$ models, which is substantially more scalable (see Table 3). Our novel attention-based search algorithm of Algorithm 2 further allows us to reduce this to training just a single model. Thus, compared to AutoCross [LWZ+19], which is the previously best known algorithm for feature cross search to the best of our knowledge, our algorithm provides at least a $150\times$ improvement in the efficiency of feature cross search, as measured by the number of required model trainings, for this natural setting of parameters. Even compared to the more efficient version of AutoCross using Sequential Attention that we consider (Algorithm 1), we obtain a $15\times$ improvement in the number of model trainings.

Table 3: Resources required to search for t order k feature crosses with m base features with vocabulary size q and embedding dimension d.

Algorithm	Model Trainings	Parameters Added
AutoCross	mkt	dq^k
AutoCross + Seq. Att. + Feature Cross (Alg. 1)	kt	mdq^k
AutoCross + Seq. Att. + Tensor Proxy (Alg. 1)	kt	md^k
Simultaneous Tensor Proxy Search (Alg. 2)	1	$t(md + d^k)$

In terms of model quality, we show in Tables 4 and 5 that, in almost all cases, at least one of either the tensor proxy model or the rounded tensor proxy model discovered by Algorithm 2 outperforms all of the non-feature cross baselines considered in Tables 1 and 2, as well as our efficient implementation of AutoCross [LWZ+19] given in Algorithm 1. Thus, our experimental results show that our tensor proxy framework for feature cross search together with Algorithm 2 not only offers substantial efficiency improvements, but also finds higher quality feature crosses in many cases. Tables 4 and 5 also include comparisons to a similar algorithm that uses a proxy-based feature cross search algorithm, but uses a small neural network to serve as the feature cross proxy rather than the tensor proxies that we define in Definition 1.1. Our results show that the tensor proxies outperform the neural network proxies in almost all cases, thus demonstrating the value of using tensor proxies in our proxy-based feature cross search framework.

Table 4: AUC of feature cross algorithms. For entries without results (–), the experiments are too expensive for the computational resources available to us. (FC = Feature Cross, TP = Tensor Proxy, NP = Neural Network Proxy)

	Adult	Bank	Credit	Employee	Frappe	Avazu	Criteo
Best Baseline	**0.9097**	0.9324	**0.8635**	0.8064	**0.9819**	0.7326	0.7934
Algorithm 1, FC	0.9047	**0.9402**	0.8572	**0.8370**	0.9817	–	–
Algorithm 1, TP	0.9021	0.9304	0.8623	0.8354	**0.9837**	0.7357	0.7924
Algorithm 2, TP	**0.9098**	0.9366	**0.8642**	0.7988	0.9823	0.7360	**0.7936**
Algorithm 2, TP, rounded	0.9045	**0.9399**	0.8620	**0.8487**	0.9814	0.7335	0.7883
Algorithm 1, NP	0.9046	0.9333	0.8626	0.8242	0.9815	**0.7368**	0.7916
Algorithm 2, NP	0.9062	0.9369	0.8637	0.8241	0.9817	0.7356	0.7944
Algorithm 2, NP, rounded	0.9055	0.9369	0.8609	0.8135	0.9772	0.7328	0.7879

Table 5: Loss feature cross algorithms. For entries without results (–), the experiments are too expensive for the computational resources available to us. (FC = Feature Cross, TP = Tensor Proxy, NP = Neural Network Proxy)

	Adult	Bank	Credit	Employee	Frappe	Avazu	Criteo
Best Baseline	**0.3131**	**0.2333**	**0.1771**	**0.2040**	0.1810	0.3951	0.4684
Algorithm 1, FC	0.3624	0.2583	0.1818	0.2161	**0.1269**	–	–
Algorithm 1, TP	0.3856	0.2766	0.1773	0.2163	0.1740	0.4072	0.4682
Algorithm 2, TP	**0.3124**	**0.2218**	**0.1763**	0.2146	0.1788	**0.3929**	0.4677
Algorithm 2, TP, rounded	0.3483	0.2457	0.1777	0.2097	**0.1241**	0.3995	0.4733
Algorithm 1, NP	0.3966	0.2471	0.1772	0.2172	0.1851	0.4017	0.4688
Algorithm 2, NP	0.3319	0.2289	0.1766	0.2118	0.1805	0.3935	**0.4676**
Algorithm 2, NP, rounded	0.3682	0.2472	0.1777	**0.2082**	0.1470	0.4002	0.4741

While the results presented previously explore a wide range of parameters and hyperparameters with a single run each, we provide a final evaluation of the improvements that we obtain over baseline algorithms using tensor proxies over multiple seeds in Figure 2. We select the best hyperparameters and baseline algorithms found by the investigations in Tables 4 and 5, and repeat the training over 10 seeds. We note that the model quality of feature cross models compares more favorably when all models are compared with the best 50% of seeds, and thus we provide this comparison as well.

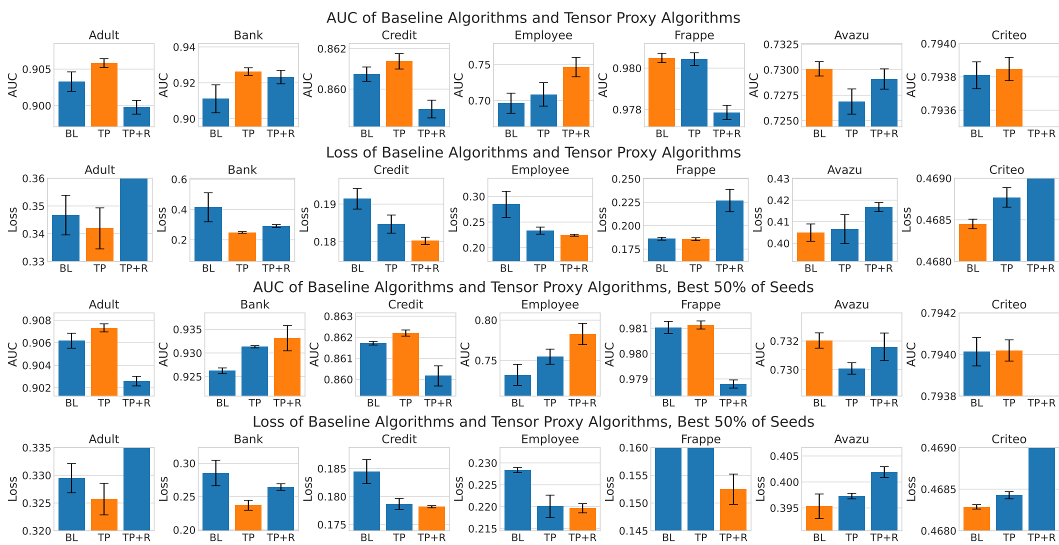

Figure 2: AUC and loss comparisons. (BL = Baseline, TP = Tensor Proxy, TP+R = TP, Rounded)

4 Conclusion

In this work, we propose an efficient feature cross search algorithm inspired by a novel connection between tensor decompositions and feature crosses, which we call tensor proxy features. We first propose a natural surrogate-based framework for feature cross search, in which we use proxies for feature cross that are computed only as a function of the feature embeddings of the original base features. Next, we introduce tensor proxy features (Definition 1.1), a specific instantiation of the feature cross proxy framework that uses a Tucker decomposition-like architecture to model higher-order feature interactions. Finally, we propose a novel search algorithm inspired by the attention mechanism (Algorithm 2), which discovers t features crosses of order k among m base features in a single model training, which substantially improves over prior greedy methods that required training mkt models. Our empirical evaluations demonstrate that in addition to being efficient, our techniques allow us to discover feature crosses and feature cross proxies that outperform all considered benchmark algorithms in many cases.

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ADHV19] Alexandr Andoni, Rishabh Dudeja, Daniel Hsu, and Kiran Vodrahalli. Attribute-efficient learning of monomials over highly-correlated variables. In Aurélien Garivier and Satyen Kale, editors, *Algorithmic Learning Theory, ALT 2019, 22-24 March 2019, Chicago, Illinois, USA*, volume 98 of *Proceedings of Machine Learning Research*, pages 127–161. PMLR, 2019.

[APVZ14a] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning polynomials with neural networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1908–1916. JMLR.org, 2014.

[APVZ14b] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning sparse polynomial functions. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 500–510. SIAM, 2014.

[BNS16] Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Global optimality of local search for low rank matrix recovery. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3873–3881, 2016.

[CEF+21] Lin Chen, Hossein Esfandiari, Gang Fu, Vahab S. Mirrokni, and Qian Yu. Feature Cross Search via Submodular Optimization. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[CSH20] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. Adaptive factorization network: Learning adaptive-order feature interactions. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3609–3616. AAAI Press, 2020.

[CWL+21] Bo Chen, Yichao Wang, Zhirong Liu, Ruiming Tang, Wei Guo, Hongkun Zheng, Weiwei Yao, Muyu Zhang, and Xiuqiang He. Enhancing explicit and implicit feature interactions via information sharing for parallel deep CTR models. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3757–3766. ACM, 2021.

[DG17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[FFG22] Matthew Fahrbach, Gang Fu, and Mehrdad Ghadiri. Subquadratic Kronecker regression with applications to tensor decomposition. *Advances in Neural Information Processing Systems*, 35:28776–28789, 2022.

[GFFM23] Mehrdad Ghadiri, Matthew Fahrbach, Gang Fu, and Vahab Mirrokni. Approximately optimal core shapes for tensor decompositions. *arXiv preprint arXiv:2302.03886*, 2023.

[GJZ17] Rong Ge, Chi Jin, and Yi Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1233–1242. PMLR, 2017.

[GTY+17] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1725–1731. ijcai.org, 2017.

[KKP+21] Arinbjörn Kolbeinsson, Jean Kossaifi, Yannis Panagakis, Adrian Bulat, Animashree Anandkumar, Ioanna Tzoulaki, and Paul M. Matthews. Tensor dropout for robust learning. *IEEE J. Sel. Top. Signal Process.*, 15(3):630–640, 2021.

[KLK+20] Jean Kossaifi, Zachary C. Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *J. Mach. Learn. Res.*, 21:123:1–123:21, 2020.

[LLZ20] Zhaocheng Liu, Qiang Liu, and Haoli Zhang. Automatically learning feature crossing from model interpretation for tabular data, 2020.

[LWZ+19] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 1936–1945. ACM, 2019.

[LZZ+18] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1754–1763. ACM, 2018.

[MSM+21] Anuj Mahajan, Mikayel Samvelyan, Lei Mao, Viktor Makoviychuk, Animesh Garg, Jean Kossaifi, Shimon Whiteson, Yuke Zhu, and Animashree Anandkumar. Tesseract: Tensorised actors for multi-agent reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7301–7312. PMLR, 2021.

[MWZ22] Arvind V. Mahankali, David P. Woodruff, and Ziyu Zhang. Low rank approximation for general tensor networks. *CoRR*, abs/2207.07417, 2022.

[NMS+19] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[Ren10] Steffen Rendle. Factorization machines. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 995–1000. IEEE Computer Society, 2010.

[SBK+20] Jiahao Su, Wonmin Byeon, Jean Kossaifi, Furong Huang, Jan Kautz, and Anima Anandkumar. Convolutional tensor-train LSTM for spatio-temporal learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien

Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[SSX+19] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In Wenwu Zhu, Dacheng Tao, Xueqi Cheng, Peng Cui, Elke A. Rundensteiner, David Carmel, Qi He, and Jeffrey Xu Yu, editors, *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 1161–1170. ACM, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[WDL+09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120, 2009.

[WFFW17] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17, Halifax, NS, Canada, August 13 - 17, 2017*, pages 12:1–12:7. ACM, 2017.

[WSC+21] Ruoxi Wang, Rakesh Shivanna, Derek Zhiyuan Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. DCN V2: improved deep & cross network and practical lessons for web-scale learning to rank systems. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1785–1797. ACM / IW3C2, 2021.

[XYH+17] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3119–3125. ijcai.org, 2017.

[YBC+23] Taisuke Yasuda, Mohammadhossein Bateni, Lin Chen, Matthew Fahrbach, Gang Fu, and Vahab Mirrokni. Sequential attention for feature selection. In *The Eleventh International Conference on Learning Representations*, 2023.

A Additional experimental details

All experiments were implemented using the TensorFlow framework [AAB+15], and the code is available at https://anonymous.4open.science/r/tensor-proxy-2847/.

Table 6: Datasets used in experiments. For datasets with an asterisk, only the first 10% of the data is used, so the original dataset is 10 times larger.

Dataset	# training examples	# features
Adult	28,000	14
Bank	33,000	10
Credit	120,000	10
Employee	26,000	9
Frappe	200,000	10
Avazu	2,800,000*	22
Criteo	3,300,000*	39

Datasets and data splits. The three datasets Frappe, Avazu, and Criteo, were obtained from preprocessed versions uploaded by the experiment implementations of [CSH20], provided at the link https://github.com/WeiyuCheng/AFN-AAAI-20 and links referenced therein. Splits for training data, validation data, and testing data are provided by the authors. The Adult dataset was obtained from the UCI machine learning repository [DG17], provided at the link https://archive.ics.uci.edu/ml/datasets/adult. The Adult dataset provides a split between training and test data, so we split the training data into training and validation data using `random.sample` in Python with a fixed seed of 2023, with $1/8$ of the training data being reserved for the validation data. The Bank, Credit, and Employee datasets were obtained from Kaggle, provided at the following links:

- Bank: https://www.kaggle.com/datasets/brijbhushannanda1979/bank-data
- Credit: https://www.kaggle.com/c/GiveMeSomeCredit/data
- Employee: https://www.kaggle.com/c/amazon-employee-access-challenge/data

These datasets do not have splits, so we again use `random.sample` in Python with a seed of 2023 to randomly split the data into training data, validation data, and testing data as an 80-10-10 split.

Base MLP model architecture. All of the baseline algorithms that we consider in this work are based around a base MLP model with an embedding layer, with additional modifications built on top of this model. In all experiments, we use a three layer MLP with 400 neurons each. All numeric features are discretized into buckets with exponentially increasing/decreasing boundaries, so all inputs can be considered to be categorical. These categorical features are then embedded into an embedding dimension of 10. When we form feature crosses, we hash the resulting vocabulary into 10^7 buckets using `tf.keras.layers.HashedCrossing`, and also embed these features into 10 dimensions. The MLP consists of a batch normalization layer and a dropout layer between each of the layers, and uses ReLU activations in the hidden layers and a sigmoid activation for the final layer.

Training. We use the Adam optimizer and a binary cross entropy loss. The learning rate is reduced on an AUC plateau in the validation data using `tf.keras.callbacks.ReduceLROnPlateau`.

Hyperparameter tuning. We tune all models with a grid search over the learning rate $\in \{10^{-2}, 10^{-3}\}$, embedding regularizer $\in \{5 \cdot 10^{-2}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$, learning rate reduction factor for `tf.keras.callbacks.ReduceLROnPlateau` $\in \{0.1, 0.15, 0.2, 0.3\}$, and dropout rate $\in \{0.1, 0.3\}$.

Compute. We run our experiments on CPU.

Variations on Algorithm 2. In addition to the basic version of our Simultaneous Tensor Proxy search algorithm in Algorithm 2, we additionally consider a number of possible modifications which may improve performance in certain cases:

- **Attention activation**: Inspired by suggestions in [YBC+23], we consider the replacement of the $\mathrm{softmax}$ activation in Algorithm 2 by other possible activation functions, including signed softmax, ℓ_1 normalization, and no activation.
- **Bias**: We consider adding an extra trainable bias to the attention-weighted features in Line 4.
- **Sequential selection**: Inspired by the Sequential Attention algorithm of [YBC+23], we additionally consider the sequential selection of t feature crosses. Unlike the findings of [YBC+23], we do not find a consistent improvement in model quality, although it does seem to help in some cases.

For all of these additional degrees of freedom, we do not find a clear pattern for when certain choices improve the performance of our algorithm, and treat these as additional hyperparameters to be tuned.

A.1 Results over multiple seeds

We provide the numbers used to generate Figures 2 below, in Tables 7 and 8.

Table 7: AUC and Losses used to generate Figure 2.

	Adult	Bank	Credit	Employee	Frappe	Avazu	Criteo
Baseline AUC	0.9033 (0.0013)	0.9111 (0.0078)	0.8607 (0.0004)	0.6963 (0.0139)	0.9805 (0.0002)	0.7301 (0.0007)	0.7938 (0.0001)
Baseline Loss	0.3467 (0.0071)	0.4148 (0.0958)	0.1914 (0.0027)	0.2845 (0.0256)	0.1860 (0.0015)	0.4049 (0.0040)	0.4685 (0.0001)
TP AUC	0.9058 (0.0006)	0.9263 (0.0021)	0.8614 (0.0004)	0.7086 (0.0163)	0.9804 (0.0003)	0.7269 (0.0012)	0.7938 (0.0001)
TP Loss	0.3419 (0.0074)	0.2490 (0.0056)	0.1847 (0.0024)	0.2332 (0.0069)	0.1856 (0.0015)	0.4066 (0.0067)	0.4688 (0.0001)
TP, Rounded AUC	0.8998 (0.0009)	0.9232 (0.0038)	0.8509 (0.0004)	0.7464 (0.0134)	0.9779 (0.0003)	0.7291 (0.0010)	0.7882 (0.0001)
TP, Rounded Loss	0.4105 (0.0077)	0.2912 (0.0096)	0.1802 (0.0010)	0.2242 (0.0019)	0.2268 (0.0119)	0.4168 (0.0021)	0.4737 (0.0002)

Table 8: AUC and Losses used to generate Figure 2, best 50% of seeds.

	Adult	Bank	Credit	Employee	Frappe	Avazu	Criteo
Baseline AUC	0.9062 (0.0007)	0.9262 (0.0006)	0.8617 (0.0001)	0.7319 (0.0131)	0.9810 (0.0002)	0.7320 (0.0005)	0.7940 (0.0001)
Baseline Loss	0.3295 (0.0026)	0.2854 (0.0191)	0.1845 (0.0021)	0.2284 (0.0006)	0.1820 (0.0010)	0.3953 (0.0024)	0.4683 (0.0000)
TP AUC	0.9073 (0.0004)	0.9313 (0.0002)	0.8622 (0.0001)	0.7546 (0.0094)	0.9811 (0.0002)	0.7301 (0.0004)	0.7940 (0.0001)
TP Loss	0.3257 (0.0028)	0.2370 (0.0075)	0.1787 (0.0010)	0.2201 (0.0026)	0.1815 (0.0009)	0.3973 (0.0005)	0.4684 (0.0000)
TP, Rounded AUC	0.9026 (0.0004)	0.9331 (0.0027)	0.8602 (0.0005)	0.7827 (0.0132)	0.9788 (0.0002)	0.7316 (0.0010)	0.7883 (0.0001)
TP, Rounded Loss	0.3906 (0.0077)	0.2644 (0.0049)	0.1782 (0.0002)	0.2197 (0.0011)	0.1525 (0.0027)	0.4019 (0.0010)	0.4732 (0.0002)