DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems

Ruoxi Wang
Google Inc.
USA
ruoxi@google.com

Rakesh Shivanna
Google Inc.
USA
rakeshshivanna@google.com

Derek Z. Cheng
Google Inc.
USA
zcheng@google.com

Sagar Jain
Google Inc.
USA
sagarj@google.com

Dong Lin
Google Inc.
USA
dongl@google.com

Lichan Hong
Google Inc.
USA
lichan@google.com

Ed H. Chi
Google Inc.
USA
edchi@google.com

Abstract

Learning effective feature crosses is the key behind building recommender systems. However, the sparse and large feature space requires exhaustive search to identify effective crosses. Deep & Cross Network (DCN) was proposed to automatically and efficiently learn bounded-degree predictive feature interactions. Unfortunately, in models that serve web-scale traffic with billions of training examples, DCN showed limited expressiveness in its cross network at learning more predictive feature interactions. Despite significant research progress made, many deep learning models in production still rely on traditional feed-forward neural networks to learn feature crosses inefficiently.

In light of the pros/cons of DCN and existing feature interaction learning approaches, we propose an improved framework DCN-V2 to make DCN more practical in large-scale industrial settings. In a comprehensive experimental study with extensive hyper-parameter search and model tuning, we observed that DCN-V2 approaches outperform all the state-of-the-art algorithms on popular benchmark datasets. The improved DCN-V2 is more expressive yet remains cost efficient at feature interaction learning, especially when coupled with a mixture of low-rank architecture. DCN-V2 is simple, can be easily adopted as building blocks, and has delivered significant offline accuracy and online business metrics gains across many web-scale learning to rank systems at Google. Our code and tutorial are open-sourced as part of TensorFlow Recommenders (TFRS)[1].

[1]Github: https://github.com/tensorflow/recommenders/blob/main/tensorflow_recommenders/layers/dcn.py; Tutorial: https://www.tensorflow.org/recommenders/examples/dcn

CCS Concepts

• **Computing methodologies → Neural networks**; *Supervised learning by classification.*

Keywords

Neural Networks, Deep Learning, Feature Crossing, CTR Prediction

ACM Reference Format:
Ruoxi Wang, Rakesh Shivanna, Derek Z. Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi . 2021. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3442381.3450078

1 Introduction

Learning to rank (LTR) [4, 27] has remained to be one of the most important problems in modern-day machine learning and deep learning. It has a wide range of applications in search, recommendation systems [17, 39, 41], and computational advertising [2, 3]. Among the crucial components of LTR models, learning effective feature crosses continues to attract lots of attention from both academia [26, 35, 46] and industry [1, 6, 13, 34, 50].

Effective feature crosses are crucial to the success of many models. They provide additional interaction information beyond individual features. For example, the combination of "`country`" and "`language`" is more informative than either one of them. In the era of linear models, ML practitioners rely on manually identifying such feature crosses [43] to increase model's expressiveness. Unfortunately, this involves a combinatorial search space, which is large and sparse in web-scale applications where the data is mostly categorical. Searching in such setting is exhaustive, often requires domain expertise, and makes the model harder to generalize.

Later on, embedding techniques have been widely adopted to project features from high-dimensional sparse vectors to much lower-dimensional dense vectors. Factorization Machines (FMs) [36, 37] leverage the embedding techniques and construct pairwise feature interactions via the inner-product of two latent vectors.

Compared to those traditional feature crosses in linear models, FM brings more generalization capabilities.

In the last decade, with more computing firepower and huge scale of data, LTR models in industry have gradually migrated from linear models and FM-based models to deep neural networks (DNN). This has significantly improved model performance for search and recommendation systems across the board [6, 13, 50]. People generally consider DNNs as universal function approximators, that could potentially learn all kinds of feature interactions [31, 47, 49]. However, recent studies [1, 50] found that DNNs are inefficient to even approximately model 2nd or 3rd-order feature crosses.

To capture effective feature crosses more accurately, a common remedy is to further increase model capacity through wider or deeper networks. This naturally crafts a double edged sword that we are improving model performance while making models much slower to serve. In many production settings, these models are handling extremely high QPS, thus have very strict latency requirements for real-time inference. Possibly, the serving systems are already pushed to a stretch that cannot afford even larger models. Furthermore, deeper models often introduce trainability issues, making models harder to train.

This has shed light on critical needs to design a model that can efficiently and effectively learn predictive feature interactions, especially in a resource-constraint environment that handles real-time traffic from billions of users. Many recent works [1, 6, 13, 26, 34, 35, 46, 50] tried to tackle this challenge. The common theme is to leverage those *implicit* high-order crosses learned from DNNs, with *explicit* and bounded-degree feature crosses which have been found to be effective in linear models. *Implicit* cross means the interaction is learned through an end-to-end function without any explicit formula modeling such cross. *Explicit* cross, on the other hand, is modeled by an explicit formula with controllable interaction order. We defer a detailed discussion of these models in Sec. 2.

Among these, Deep & Cross Network (DCN) [50] is effective and elegant, however, productionizing DCN in large-scale industry systems faces many challenges. The expressiveness of its cross network is limited. The polynomial class reproduced by the cross network is only characterized by O(input size) parameters, largely limiting its flexibility in modeling random cross patterns. Moreover, the allocated capacity between the cross network and DNN is unbalanced. This gap significantly increases when applying DCN to large-scale production data. An overwhelming portion of the parameters will be used to learn implicit crosses in the DNN.

In this paper, we propose a new model *DCN-V2* that improves the original DCN model. We have already successfully deployed DCN-V2 in quite a few learning to rank systems across Google with significant gains in both offline model accuracy and online business metrics. DCN-V2 first learns explicit feature interactions of the inputs (typically the embedding layer) through cross layers, and then combines with a deep network to learn complementary implicit interactions. The core of DCN-V2 is the cross layers, which inherit the simple structure of the cross network from DCN, however significantly more expressive at learning explicit and bounded-degree cross features. The paper studies datasets with clicks as positive labels, however DCN-V2 is label agnostic and can be applied to any learning to rank systems. The main contributions of the paper are five-fold:

- We propose a novel model—DCN-V2—to learn effective explicit and implicit feature crosses. Compared to existing methods, our model is more expressive yet remains efficient and simple.
- Observing the low-rank nature of the learned matrix in DCN-V2, we propose to leverage low-rank techniques to approximate feature crosses in a subspace for better performance and latency trade-offs. In addition, we propose a technique based on the Mixture-of-Expert architecture [19, 45] to further decompose the matrix into multiple smaller sub-spaces. These sub-spaces are then aggregated through a gating mechanism.
- We conduct and provide an extensive study using synthetic datasets, which demonstrates the inefficiency of traditional ReLU-based neural nets to learn high-order feature crosses.
- Through comprehensive experimental analysis, we demonstrate that our proposed DCN-V2 models significantly outperform SOTA algorithms on Criteo and MovieLen-1M benchmark datasets.
- We provide a case study and share lessons in productionizing DCN-V2 in a large-scale industrial ranking system, which delivered significant offline and online gains.

Our code and tutorial are open-sourced as part of TensorFlow Recommenders (TFRS)[2].

2 Related Work

The core idea of recent feature interaction learning work is to leverage both explicit and implicit (from DNNs) feature crosses. To model explicit crosses, most recent work introduces multiplicative operations ($x_1 \times x_2$) which is inefficient in DNN, and designs a function $f(\mathbf{x}_1, \mathbf{x}_2)$ to efficiently and explicitly model the pairwise interactions between features $\mathbf{x}_1$ and $\mathbf{x}_2$. We organize the work in terms of how they combine the explicit and implicit components.

Parallel Structure. One line of work jointly trains two parallel networks inspired from the wide and deep model [6], where the wide component takes inputs as crosses of raw features; and the deep component is a DNN model. However, selecting cross features for the wide component falls back to the feature engineering problem for linear models. Nonetheless, the wide and deep model has inspired many works to adopt this parallel architecture and improve upon the wide component.

DeepFM [13] automates the feature interaction learning in the wide component by adopting a FM model. DCN [50] introduces a cross network, which learns explicit and bounded-degree feature interactions automatically and efficiently. xDeepFM [26] increases the expressiveness of DCN by generating multiple feature maps, each encoding all the pairwise interactions between features at current level and the input level. Besides, it also considers each feature embedding $\mathbf{x}_i$ as a unit instead of each element x_i as a unit. Unfortunately, its computational cost is significantly high (10x of #params), making it impractical for industrial-scale applications. Moreover, both DeepFM and xDeepFM require all the feature embeddings to be of equal size, yet another limitation when applying to industrial data where the vocab sizes (sizes of categorical features) vary from

[2]Github: https://github.com/tensorflow/recommenders/blob/main/tensorflow_recommenders/layers/dcn.py; Tutorial: https://www.tensorflow.org/recommenders/examples/dcn

$O(10)$ to millions. AFM [52] assigns importance to each feature interaction through an attention network. AutoInt [46] leverages the multi-head self-attention mechanism with residual connections; InterHAt [25] further employs Hierarchical Attentions.

Stacked Structure. Another line of work introduces an interaction layer—which creates explicit feature crosses—in between the embedding layer and a DNN model. This interaction layer captures feature interaction at an early stage, and facilitates the learning of subsequent hidden layers. Product-based neural network (PNN) [35] introduces inner (IPNN) and outer (OPNN) product layer as the pairwise interaction layers. One downside of OPNN lies in its high computational cost. Neural FM (NFM) [16] extends FM by replacing the inner-product with a Hadamard product; DLRM [34] follows FM to compute the feature crosses through inner products; These models can only create up to 2nd-order explicit crosses. AFN [7] transforms features into a log space and adaptively learns arbitrary-order feature interactions. Similar to DeepFM and xDeepFM, these methods only accept embeddings of equal sizes.

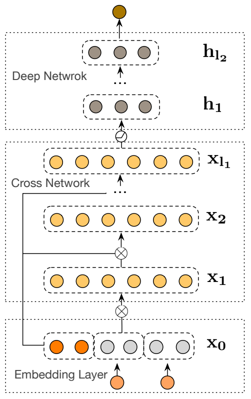
Despite many advances made, our comprehensive experiments (Sec. 7) demonstrate that DCN still remains to be a strong baseline. We attribute this to its simple structure that has facilitated the optimization. However, as discussed, its limited expressiveness has prevented it from learning more effective feature crosses in web-scale systems. In the following, we present a new architecture that inherits DCN's simple structure while increasing its expressiveness.

3 Proposed Architecture: DCN-V2

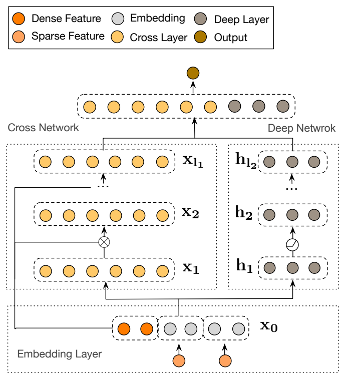
This section describes a novel model architecture — DCN-V2 — to learn both explicit and implicit feature interactions. DCN-V2 starts with an *embedding layer*, followed by a *cross network* containing multiple cross layers that models explicit feature interactions, and then combines with a *deep network* that models implicit feature interactions. The improvements made in DCN-V2 are **critical for putting DCN into practice for highly-optimized production systems**. DCN-V2 significantly improves the expressiveness of DCN [50] in modeling complex explicit cross terms in web-scale production data, while maintaining its elegant formula for easy deployment. The function class modeled by DCN-V2 is a strict superset of that modeled by DCN. The overall model architecture is depicted in Fig. 1, with two ways to combine the cross network with the deep network: (1) stacked and (2) parallel. In addition, observing the low-rank nature of the cross layers, we propose to leverage a mixture of low-rank cross layers to achieve healthier trade-off between model performance and efficiency.

3.1 Embedding Layer

The embedding layer takes as input a combination of categorical (sparse) and dense features, and outputs $\mathbf{x}_0 \in \mathbb{R}^d$. It follows a similar setup as [50]. Unlike many related works [13, 16, 26, 34, 35, 46] which requires embedding sizes to match, our model accepts arbitrary embedding sizes. This is particularly important for industrial recommenders where the vocab size varies from $O(10)$ to $O(10^8)$.



(a) Stacked



(b) Parallel

Figure 1: Visualization of DCN-V2. $\otimes$ represents the cross operation in Eq. (1), *i.e.*, $\mathbf{x}_{l+1} = \mathbf{x}_0 \odot (W_l \mathbf{x}_l + \mathbf{b}_l) + \mathbf{x}_l$.

3.2 Cross Network

The core of DCN-V2 lies in the cross layers that create explicit feature crosses. Eq. (1) shows the $(l+1)^{\text{th}}$ cross layer.

$$\mathbf{x}_{l+1} = \mathbf{x}_0 \odot (W_l \mathbf{x}_l + \mathbf{b}_l) + \mathbf{x}_l \tag{1}$$

where $\mathbf{x}_0 \in \mathbb{R}^d$ is the base layer that contains the original features of order 1, and is normally set as the embedding layer. $\mathbf{x}_l, \mathbf{x}_{l+1} \in \mathbb{R}^d$, respectively, represents the input and output of the $(l+1)^{\text{th}}$ cross layer. $W_l \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_l \in \mathbb{R}^d$ are the learned weight matrix and bias vector. Fig. 2 visualizes an individual cross layer.

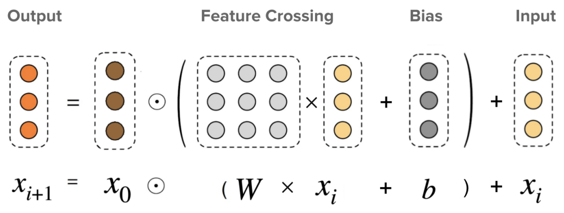


Figure 2: Visualization of a cross layer.

For an l-layered cross network, the highest polynomial order is $l+1$ and the network contains all the feature crosses up to the highest order. Please see Sec. 4 for a detailed analysis, both from bitwise and feature-wise point of views. When $W = \mathbf{1} \times \mathbf{w}^\top$, where $\mathbf{1}$ represents a vector of ones, DCN-V2 falls back to DCN.

The cross layers could only reproduce polynomial classes of bounded degree; and any other complex function space could only be approximated[3]. Hence, we introduce a deep network next to complement the modeling of the inherent distribution in the data.

3.3 Deep Network

The l^{th} deep layer's formula is given by $\mathbf{h}_{l+1} = f(W_l \mathbf{h}_l + \mathbf{b}_l)$, where $\mathbf{h}_l \in \mathbb{R}^{d_l}, \mathbf{h}_{l+1} \in \mathbb{R}^{d_{l+1}}$, are the input and output of the l-th deep layer. $W_l \in \mathbb{R}^{d_l \times d_{l+1}}$ is the weight matrix and $\mathbf{b}_l \in \mathbb{R}^{d_{l+1}}$ is the bias vector. $f(\cdot)$ is an elementwise activation function and we set it to be ReLU, although any other activation functions are also suitable.

[3]Any function with certain smoothness assumptions can be well-approximated by polynomials. In fact, we've observed in our experiments that cross network alone was able to achieve similar performance as traditional deep networks.

3.4 Deep and Cross Combination

We seek structures to combine the cross network and deep network. Recent literature adopted two structures: stacked and parallel. In practice, we have found that which architecture works better is data dependent. Hence, we present both:

Stacked Structure (Fig. 1a): The input $\mathbf{x}_0$ is fed to the cross network followed by the deep network, and the final layer is given by $\mathbf{x}_{\text{final}} = \mathbf{h}_{L_d}$, $\mathbf{h}_0 = \mathbf{x}_{L_c}$, which models the data as $f_{\text{deep}} \circ f_{\text{cross}}$.

Parallel Structure (Fig. 1b): The input $\mathbf{x}_0$ is fed in parallel to both the cross and deep networks. The outputs $\mathbf{x}_{L_c}$ and $\mathbf{h}_{L_d}$ are concatenated to create the final output layer $\mathbf{x}_{\text{final}} = [\mathbf{x}_{L_c}; \mathbf{h}_{L_d}]$. This structure models the data as $f_{\text{cross}} + f_{\text{deep}}$.

In the end, the prediction is $\hat{y}_i := \sigma(\mathbf{w}_{\text{logit}}^\top \mathbf{x}_{\text{final}})$, where $\mathbf{w}_{\text{logit}}$ is the weight vector for the logit, and $\sigma(x) = 1/(1+\exp(-x))$. For the final loss, we use the Log Loss (with a regularization term), which is commonly used for learning to rank systems especially with binary labels (e.g., clicks). Note that DCN-V2 itself is both prediction-task and loss-function agnostic.

3.5 Cost-Effective Mixture of Low-Rank DCN

DCN-V2 has successfully enabled significant model quality improvements for many highly-optimized production systems. For those production models, the model capacity is often constrained by limited serving resources and strict latency requirements. Hence, we seek methods to make DCN-v2 even more cost-efficient.

The simple structure of DCN-v2, where the computational bottleneck lies in matrix-vector multiplications, has allowed us to leverage matrix approximation techniques to reduce the cost. Low-rank techniques [12] are widely used [5, 9, 14, 20, 51, 53] to reduce the computational cost. It approximates a dense matrix $M \in \mathbb{R}^{d\times d}$ by two tall and skinny matrices $U, V \in \mathbb{R}^{d\times r}$. When $r \le d/2$, the cost will be reduced. However, they are most effective when the matrix shows a large gap in singular values or a fast spectrum decay. In many settings, we indeed observe that the learned matrix is numerically low-rank in practice.

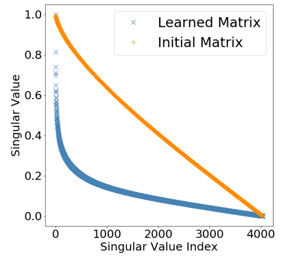
Fig. 3a shows the singular decay pattern of the learned matrix W in DCN-V2 (see Eq. (1)) from a production model. Compared to the initial matrix, the learned matrix shows a much faster spectrum decay pattern. Let's define the numerical rank R_T with tolerance T to be $\text{argmin}_k(\sigma_k < T \cdot \sigma_1)$, where $\sigma_1 \ge \sigma_2 \ge, \ldots, \ge \sigma_n$ are the singular values. Then, R_T means majority of the mass up to tolerance T, is preserved in the top k singular values. In the field of machine learning and deep learning, a model could still work surprisingly well with a reasonably high tolerance T [4].

Hence, it is well-motivated to impose a low-rank structure on W. Eq (2) shows the resulting $(l+1)$-th low-rank cross layer

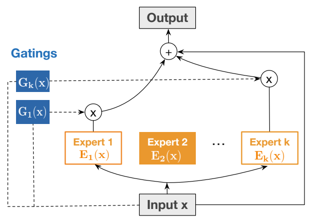
$$\mathbf{x}_{l+1} = \mathbf{x}_0 \odot \left(U_l\left(V_l^\top \mathbf{x}_i\right) + \mathbf{b}_l\right) + \mathbf{x}_i \tag{2}$$

where $U_l, V_l \in \mathbb{R}^{d\times r}$ and $r \ll d$. Eq (2) has two *interpretations*: 1) we learn feature crosses in a subspace; 2) we project the input $\mathbf{x}$ to lower-dimensional $\mathbb{R}^r$, and then project it back to $\mathbb{R}^d$. The interpretations have inspired the following two model improvements.

[4]This is very different from the filed of scientific computing (*e.g.*, solving linear systems), where the approximation accuracy need to be very high. For problems such as CTR prediction, some errors could introduce regularization effect to the model.



(a) Singular Values



(b) Mixture of Low-rank Experts

Figure 3: Left: Normalized singular values ($1 = \sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_k$) of the learned DCN-V2 weight matrix. Right: Visualization of mixture of low-rank cross layer.

Interpretation 1 inspires us to adopt the idea from Mixture-of-Experts (MoE) [10, 19, 30, 45]. MoE-based models consist of two components: experts (typically a small network) and gating (a function of inputs). In our case, instead of relying on one single expert (Eq (2)) to learn feature crosses, we leverage multiple such experts, each learning feature interactions in a different subspaces, and adaptively combine the learned crosses using a gating mechanism that depends on input $\mathbf{x}$. The resulting mixture of low-rank cross layer formulation is shown in Eq. (3) and depicted in Fig. 3b.

$$\begin{aligned}\mathbf{x}_{l+1} &= \sum_{i=1}^{K} G_i(\mathbf{x}_l) E_i(\mathbf{x}_l) + \mathbf{x}_l \\ E_i(\mathbf{x}_l) &= \mathbf{x}_0 \odot \left(U_l^i\left(V_l^{i\top}\mathbf{x}_l\right) + \mathbf{b}_l\right)\end{aligned} \tag{3}$$

where K is the number of experts; $G_i(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}$ is the gating function, common sigmoid or softmax; $E_i(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^d$ is the i^{th} expert in learning feature crosses. $G(\cdot)$ dynamically weights each expert for input $\mathbf{x}$, and when $G(\cdot) \equiv 1$, Eq (3) falls back to Eq (2).

Interpretation 2 inspires us to leverage the low-dimensional nature of the projected space. Instead of immediately projecting back from $\mathbb{R}^{d'}$ to $\mathbb{R}^d$ ($d' \ll d$), we further apply nonlinear transformations in the projected space to refine the representation [11].

$$E_i(\mathbf{x}_l) = \mathbf{x}_0 \odot \left(U_l^i \cdot g\left(C_l^i \cdot g\left(V_l^{i\top}\mathbf{x}_l\right)\right) + \mathbf{b}_l\right) \tag{4}$$

where $g(\cdot)$ represents any nonlinear activation function.

Discussions. This section aims to make effective use of the fixed memory/time budget to learn meaningful feature crosses. From Eqs (1)–(4), each formula represents a strictly larger function class assuming a fixed #params. Instead of post-training compression, our model imposes the structure prior to training and jointly learn the parameters. Due to that, the cross layer is an integral part of the nonlinear system $f(\mathbf{x}) = \left(f_k(W_k) \circ \cdots \circ f_1(W_1)\right)(\mathbf{x})$, where $(f_{i+1} \circ f_i)(\cdot) := f_{i+1}(f_i(\cdot))$. Hence, the training dynamics of the overall system might be affected, and it would be interesting to study how the global statistics, such as Jacobian and Hession matrices of $f(\mathbf{x})$, are affected. We leave such investigations to future work.

Complexity Analysis. Let d denote the embedding size, L_c denote the number of cross layers, K denote the number of low-rank DCN experts. Further, we assume each expert has the same smaller dimension r (upper bound on the rank). The time and space complexity for the cross network is $O(d^2 L_c)$, and for mixture of low-rank DCN (mixDCN) it's efficient when $rK \ll d$ with $O(2drKL_c)$.

4 Model Analysis

This section analyzes DCN-V2 from the perspective of polynomial approximation. We adopt notations from [50].

Notations. Let the embedding vector $\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_k] = [x_1, x_2, \ldots, x_d] \in \mathbb{R}^d$ be a column vector, where $\mathbf{x}_i \in \mathbb{R}^{e_i}$ represents the i-th feature embedding, and x_i represents the i-th element in $\mathbf{x}$. Let multi-index $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_d] \in \mathbb{N}^d$ and $|\boldsymbol{\alpha}| = \sum_{i=1}^d \alpha_i$. $C_a^b := \left\{\mathbf{y} \in \{1, \cdots, a\}^b \;\middle|\; \forall i < j, y_i > y_j\right\}$. Let $\mathbf{1}$ be a vector of all 1's. We use capital letters for matrices, bold lower-case letters for vectors, and normal lower-case letters for scalars.

Bitwise-Interactions. Consider each bit x_i as a unit, then, the output of an l-layered cross network reproduces polynomials in:

$$\left\{\sum\nolimits_{\boldsymbol{\alpha}} c_{\boldsymbol{\alpha}}\left(W^{(1)}, \ldots, W^{(l)}\right) x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d} \;\middle|\; 0 \le |\boldsymbol{\alpha}| \le l{+}1, \boldsymbol{\alpha} \in \mathbb{N}^d\right\},$$

where $c_{\boldsymbol{\alpha}} = \sum_{\mathbf{j} \in C_l^{|\boldsymbol{\alpha}|-1}} \sum_{\mathbf{i} \in P_{\boldsymbol{\alpha}}} \prod_{k=1}^{|\boldsymbol{\alpha}|-1} w_{i_k i_{k+1}}^{(j_k)}$, $w_{ij}^{(k)}$ is the $(i,j)^{\text{th}}$ entry of $W^{(k)}$, and $P_{\boldsymbol{\alpha}} = \text{Permutations}\,(\cup_i \{\underbrace{i, \ldots, i}_{\alpha_i \text{ times}} \mid \alpha_i \neq 0\})$.

Feature-wise Interactions. Consider each feature $\mathbf{x}_i$ as a unit, then, the output layer of an l-layered cross network consists of all the feature interactions up to order $l + 1$. Specifically, for $\mathbf{x}_i$'s with their (repeated) indices in I, their order-p interaction is given by:

$$\sum\nolimits_{\mathbf{i} \in P_I} \sum\nolimits_{\mathbf{j} \in C_p^{p-1}} \mathbf{x}_{i_1} \odot \left(W_{i_1, i_2}^{(j_1)} \mathbf{x}_{i_2} \odot \ldots \odot \left(W_{i_l, i_{l+1}}^{(j_l)} \mathbf{x}_{i_{l+1}}\right)\right)$$

where P_I = Permutations(I). The proofs are in the Appendix.

From both bitwise and feature-wise perspectives, the cross network is able to create all the feature interactions up to order $l + 1$ for an l-layered cross network. Compared to DCN, DCN-V2 characterizes the same polynomial class with more parameters and is more expressive. Moreover, the feature interactions in DCN-V2 is more expressive and can be viewed both bitwise and feature-wise, whereas in DCN it is only bitwise [26, 46, 50].

5 Research Questions

We seek answers for these following research questions:

RQ1 When would feature cross learning methods become more efficient than ReLU-based DNNs?

RQ2 How does the proposed DCN-V2 compare to SOTA methods? Could we achieve healthier trade-off between model accuracy and cost through DCN-V2 and mixture of low-rank DCN?

RQ3 How does the settings in DCN-V2 affect model quality?

RQ4 Is DCN-V2 capturing important feature crosses? Does the model provide good understandability?

Throughout the paper, "CrossNet" or "CN" represents the cross network; suffix "Mix" denotes the mixture of low-rank version.

6 Empirical understanding of feature crossing techniques (RQ1)

Many recent works [1, 6, 13, 26, 34, 35, 50] proposed to model explicit feature crosses that couldn't be learned efficiently from traditional neural networks. However, most works only studied public datasets with unknown cross patterns and noisy data. Few work has studied in a clean setting with known ground-truth models. Therefore, it's important to understand: 1) in which cases would traditional neural nets become inefficient; 2) the role of each component in our proposed model DCN-V2.

We use the cross network in DCN models to represent those feature cross methods and compare with ReLUs, which are commonly used in industrial recommender systems. To simplify experiments and ease understanding, we assume each feature $x_i \in \mathbb{R}$, and monomial $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}$ represents a $|\boldsymbol{\alpha}|$-th order feature cross.

Performance with increasing difficulty. Consider only 2nd-order feature crosses and let the ground-truth model be $f(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|=2} w_{\boldsymbol{\alpha}} x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d}$. Then, the difficulty of learning $f(\mathbf{x})$ depends on: 1) sparsity ($w_{\boldsymbol{\alpha}} = 0$), the number of crosses, and 2) similarity of the cross patterns (characterized by Var$(w_{\boldsymbol{\alpha}})$), meaning a change in one feature would simultaneously affect most feature crosses by a similar amount. We create synthetic datasets with increasing difficulty as in Eq. (5).

$$\begin{aligned} f_1(\mathbf{x}) &= x_1^2 + x_1 x_2 + x_3 x_1 + x_4 x_1 \\ f_2(\mathbf{x}) &= x_1^2 + 0.1 x_1 x_2 + x_2 x_3 + 0.1 x_3^2 \\ f_3(\mathbf{x}) &= \sum\nolimits_{(i,j) \in S} w_{ij} x_i x_j, \;\; \mathbf{x} \in \mathbb{R}^{100}, |S| = 100 \end{aligned} \tag{5}$$

where set S and weights w_{ij} are randomly assigned, and x_i's are uniformly sampled from interval [-1, 1].

Tab. 1 reports mean RMSE out of 5 runs and the model size. When the cross patterns are simple (f_1), both DCN-V2 and DCN are efficient. When the patterns become more complicated (f_3), DCN-V2 remains accurate while DCN degrades. DNN's performance remains poor even with a wider and deeper structure (layer sizes [200, 200] for f_1 and f_2, [1024, 512, 256] for f_3). This suggests the inefficiency of DNN in modeling monomial patterns.

Table 1: RMSE and Model Size (# Parameters) for Polynomial Fitting of Increasing Difficulty.

	DCN (1Layer)		DCN-V2 (1Layer)		DNN (1Layer)		DNN (large)	
	RMSE	Size	RMSE	Size	RMSE	Size	RMSE	Size
f_1	8.9E-13	12	**5.1E-13**	24	2.7E-2	24	4.7E-3	41K
f_2	1.0E-01	9	**4.5E-15**	15	3.0E-2	15	1.4E-3	41K
f_3	2.6E+00	300	**6.7E-07**	10K	2.7E-1	10K	7.8E-2	758K

Role of each component. We also conducted ablation studies on homogeneous polynomials of order 3 and 4, respectively. For each order, we randomly selected 20 cross terms from $\mathbf{x} \in \mathbb{R}^{50}$.

Fig. 4 shows mean RMSE v.s. layer depth. Clearly, $\mathbf{x}_0 \odot (W\mathbf{x}_i)$ models order-d crosses at layer d-1, verified by its best performance achieved at layer 2 for 3rd-order polynomial (similar for 4th-order). At other layers, however, the performance significantly degrades. This is where the bias and residual terms are helpful — they create and maintain all the crosses up to the highest order. This reduces the performance gap between layers, and stabilizes the model when redundant crosses are introduced. This is particularly important for real-world applications with unknown cross patterns.

Fig. 4 also reveals the limited expressiveness of DCN in modeling complicated cross patterns.

To summarize, ReLUs are inefficient in capturing explicit feature crosses (multiplicative relations) even with a deeper and larger network. This is well aligned with previous studies [1]. The accuracy considerably degrades when the cross patterns become more complicated. DCN accurately captures simple cross patterns but fails

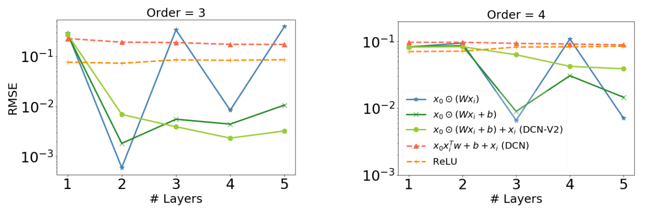


Figure 4: Homogeneous polynomial fitting of 3rd and 4th order. The legend shows DCN-V2 with different component(s).

at more complicated ones. DCN-V2, on the other hand, remains accurate and efficient for complicated cross patterns.

7 Experimental Results (RQ2 - RQ4)

This section empirically verifies the effectiveness of DCN-V2 in feature interaction learning across 3 datasets and 2 platforms, compared with SOTA. In light of recent concerns about poor reproducibility of published results [8, 33, 38], we conducted a fair and comprehensive experimental study with extensive hyper-parameter search to properly tune all the baselines and proposed approaches. In addition, for each optimal setup, we train 5 models with different random initialization, and report the mean and standard deviation.

Sec. 7.2 compares DCN-V2 with all the baselines comprehensively (**RQ2**). Sec. 7.3 evaluates the influence of hyper-parameters on the performance of DCN-V2 (**RQ3**). Sec. 7.4 focuses on model understanding (**RQ4**) of whether we are indeed discovering meaningful feature crosses with DCN-V2.

7.1 Experiment Setup

This section describes training datasets, baseline approaches, and details of the hyper-parameter search and training process.

7.1.1 Datasets We use 2 popular and standard public datasets.

Criteo[5]. This click-through-rate (CTR) prediction benchmark dataset contains user logs over a period of 7 days. It has 45M examples and 39 features. We follow [46, 50] and use first 6 days for training, and randomly split the last day's data into validation and test set equally. We log-normalize (*i.e.*, $\log(x + 4)$ for feature-2 and $\log(x+1)$ for others) the 13 dense features and embed the remaining 26 categorical features.

MovieLen-1M[6]. The data contains 740k examples and 7 features. Each training example includes a ⟨`user-features, movie-features, rating`⟩ triplet. Similar to [46], we formalize the task as a regression problem. All the ratings for 1s and 2s are normalized to be 0s; 4s and 5s to be 1s; and rating 3s are removed. 6 non-multivalent categorical features are used and embedded. The data is randomly split into 80% for training, 10% for validation and 10% for testing.

7.1.2 Baselines. We compare our proposed approaches with 6 SOTA feature interaction learning algorithms. A brief comparison between the approaches is highlighted in Tab. 2.

7.1.3 Implementation Details. All the baselines and our approaches are implemented in TensorFlow v1. For a fair comparison, all the implementations were identical across all the models except for the feature interaction component [7].

Table 2: High-level model comparisons. Assuming the input $\mathbf{x}_0 = [\mathbf{v}_1; \ldots; \mathbf{v}_k]$ contains k feature embeddings. $\oplus$:= concatenation; $\otimes$:= outer-product; $\odot$:= Hadamard-product. $f_i(\cdot)$ represents implicit feature interactions, *i.e.*, ReLU layers. In the last column, the '+' sign is on the logit level

Model	Explicit Interactions (f_e)		Final
	Order	(Simplified) Key Formula	Objective
PNN [35]	2	$\mathbf{x}_o = [\mathbf{v}_i^\top \mathbf{v}_j \mid \forall i, j]$ (IPNN) $\mathbf{x}_o = [\mathrm{vec}(\mathbf{v}_i \otimes \mathbf{v}_j) \mid \forall i, j]$ (OPNN)	$f_i \circ f_e$
DeepFM [13]	2	$\mathbf{x}_o = [\mathbf{v}_i^\top \mathbf{v}_j \mid \forall i, j]$	$f_i + f_e$
DLRM [34]	2	$\mathbf{x}_o = [\mathbf{v}_i^\top \mathbf{v}_j \mid \forall i, j]$	$f_i \circ f_e$
DCN [50]	≥ 2	$\mathbf{x}_{i+1} = \mathbf{x}_0 \otimes \mathbf{x}_i \mathbf{w}_i$	$f_i + f_e$
xDeepFM [26]	≥ 2	$\mathbf{v}_h^k = \sum_{i,j} w_{ij}^{kh} (\mathbf{v}_i^{k-1} \odot \mathbf{v}_j)$	$f_i + f_e$
AutoInt [46]	NA	$\widetilde{\mathbf{v}}_i = g\left(\frac{\sum_j \exp(\langle W_q \mathbf{v}_i, W_k \mathbf{v}_j \rangle) W_v \mathbf{v}_j}{\sum_j \exp(\langle W_q \mathbf{v}_i, W_k \mathbf{v}_j \rangle)})\right)$	$f_i + f_e$
DCN-V2 (ours)	≥ 2	$\mathbf{x}_i = \mathbf{x}_0 \odot (W_i \mathbf{x}_i)$	$f_i \circ f_e$ $f_i + f_e$

Embeddings. All the baselines require each feature's embedding size to be the same except for DNN and DCN models. Hence, we fixed it to be $\mathrm{Avg}_{\mathrm{vocab}}(6 \cdot (\text{vocab cardinality})^{\frac{1}{4}})$ (39 for Criteo and 30 for Movielen-1M) for all the models[8].

Optimization. We used Adam [22] with a batch size of 512 (128 for MovieLen). The kernels were initialized with He Normal [15], and biases to **0**; the gradient clipping norm was 10; an exponential moving average with decay 0.9999 to parameters was applied.

Reproducibility and fair comparisons: hyper-parameters tuning and results reporting. For all the baselines, we conducted a coarse-level (larger-range) grid search over the hyper-parameters, followed by a finer-level (smaller-range) search. To ensure reproducibility and mitigate model variance, for each approach and dataset, we report the mean and stddev out of 5 independent runs for the best configuration. We describe detailed settings below for Criteo; and follow a similar process for MovieLens with different ranges.

For all the baselines on Criteo, the learning rate was tuned from 10^{-4} to 10^{-1} on a log scale and then narrowed down to 10^{-4} to 5×10^{-4} on a linear scale. The training steps were in {150k, 160k, 200k, 250k, 300k}. The hidden layer depth ranged in {1, 2, 3, 4} with their layer sizes in {562, 768, 1024}. And the regularization parameter λ was in $\{0, 3 \times 10^{-5}, 10^{-4}\}$.

We then describe each model's own hyper-parameters, where the search space is designed based on reported setting. For DCN, the number of cross layers ranged from 1 to 4. For AutoInt, the number of attention layers was from 2 to 4; the attention embedding size was in {20, 32, 40}; the number of attention head was from 2 to 3; and the residual connection was either on or off. For xDeepFM, the CIN layer size was in {100, 200}, depth in {2, 3, 4}, activation was identity, computation was either direct or indirect. For DLRM, the bottom MLP layer sizes and numbers were in {(512,256,64), (256,64)}. For PNN, we ran for IPNN, OPNN and PNN*, and for the latter two, the kernel type ranged in {full matrix, vector, number}. For all the

[5]http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset
[6]https://grouplens.org/datasets/movielens

[7]We adopted implementation from https://github.com/Leavingseason/xDeepFM, https://github.com/facebookresearch/dlrm and https://github.com/shenweichen/DeepCTR
[8]This formula is a rule-of-thumb number that is widely used [50], also see https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html

models, the total number of parameters was capped at $1024^2 \times 5$ to limit the search space and avoid overly expensive computations.

7.2 Model Performance (RQ2)

This section compares the performance between DCN-V2 approaches and the baselines. Note that the best setting reported for each model was searched over a **wide-ranged model capacity and hyper-parameter space** including the baselines. And if two settings performed on-par, we report the **lower-cost** one. Tab. 3 shows the best LogLoss and AUC (Area Under the ROC Curve) on testset for Criteo and MovieLen. For Criteo, a **0.001-level improvement** is considered significant (see [13, 46, 50]).

We see that DCN-V2 consistently outperformed the baselines (including DNN) and achieved a healthy quality/cost trade-off. In our thorough hyper-parameter search for optimal settings for baseline models, we did explore wider and deeper models. However an even larger model could not yield more quality gains, clearly showing that many of the baselines were bottlenecked by quality rather than efficiency. It's also worth mentioning that the baselines' performances reported in Tab. 3 were improved over the numbers reported by previous papers (see Tab. 6a in Appendix); however, they were on-par and sometimes worse than the ReLU-based DNN with fine-granular model tuning.

Best Settings. The optimal settings are in Tab. 3. For DCN-V2 models, both the 'stacked' and 'parallel' structures outperformed all the baselines, while 'stacked' worked better on Criteo and 'parallel' worked better on Movielen-1M. On Criteo, the setting was gate as constant, hard_tanh activation for DCN-Mix; gate as softmax and identity activation for CrossNet. The best training steps was 150k for all the baselines; learning rate varies for all the models. Note that the best settings should depend on the data, model, and infrastructure. In practice, we found that 'stacked' structure improves the quality more while 'parallel' structure helps reduce the model variance.

Model Quality — Comparisons among baselines. For 2nd-order methods, DLRM performed inferiorly to DeepFM although they are both derived from FM. This might be due to DLRM's omission of the 1st-order sparse features after the dot-product layer. For higher-order methods, xDeepFM, AutoInt and DCN behaved similarly on Criteo, while on MovieLens xDeepFm showed a high variance in Logloss.

DCN-V2 achieved the best performance (0.001 considered to be significant on Criteo [26, 46, 50]) by explicitly modeling up to 3rd-order crosses beyond those implicit ones from DNN. DCN-Mix, the mixture of low-rank DCN, efficiently utilized the memory and reduced the cost by 30% while maintaining the accuracy. Interestingly, CrossNet alone outperformed DNN on both datasets; we defer more discussions to end of this section.

Model Quality — Comparisons with DNN. DNNs are universal approximators and tough-to-beat baselines when highly-optimized. Hence, we properly tuned DNN with all the baselines, and used a larger layer size than those used in literature (*e.g.*, 200 - 400 in [26, 46]). **To our surprise, DNN performed neck to neck with most baselines and even outperformed certain models.**

Our hypothesis is that those explicit feature crosses from baselines were not modeled in an **expressive** and **easy-to-optimize** manner. The former makes its performance easy to be matched by a DNN with large capacity. The latter would easily lead to trainability issues, making the model unstable, hard to identify a good local optima or to generalize. Hence, when integrated with DNN, the overall performance is dominated by the DNN component. This becomes especially true with a large-capacity DNN, which could already approximate some simple cross patterns.

In terms of expressiveness, consider the 2nd-order methods. PNN models crosses more expressively than DeepFM and DLRM, which resulted in its superior performance on MovieLen-1M. This also explains the inferior performance of DCN compared to DCN-V2.

In terms of trainability, certain models might be inherently more difficult to train and resulted in unsatisfying performance. Consider PNN. On MoiveLen-1M, it outperformed DNN, suggesting the effectiveness of those 2nd-order crosses. On Criteo, however, PNN's advantage has diminished and the averaged performance was on-par with DNN. This was caused by the instability of PNN. Although its best run was better than DNN, its high stddev from multiple trials has driven up the mean loss. xDeepFM also suffers from trainability issue (see its high stddev on MovieLens). In xDeepFM, each feature map encodes all the pair-wise crosses while only relies on a single variable to learn the importance of each cross. In practice, a single variable is difficult to be learned when jointly trained with magnitudes more parameters. Then, an improperly learned variable would lead to noises.

DCN-V2, on the other hand, consistently outperforms DNN. It successfully leveraged both the explicit and implicit feature interactions. We attribute this to the balanced number of parameters between the cross network and the deep network (**expressive**), as well as the simple structure of cross net which eased the optimization (**easy-to-optimize**). It's worth noting that the high-level structure of DCN-V2 shares a similar spirit of the self-attention mechanism adopted in AutoInt, where each feature embedding attends to a weighed combination of other features. The difference is that during the attention, higher-order interactions were modeled explicitly in DCN-V2 but implicitly in AutoInt.

Model Efficiency. Tab. 3 also provides details for model size and FLOPS. FLOPS is a close estimation of run time, which is subjective to implementation details (See Tab. **??**b in the Appendix for complexity analysis). The reported setting was properly tuned over the hyper-parameters of each model and the DNN component. For most models, the FLOPS is roughly 2x of the #params; for xDeepFM, however, it is one magnitude higher, making it impractical in industrial-scale applications (also observed in [46]). Note that for DeepFM and DLRM, we've also searched over larger-capacity models; however, they didn't deliver better quality. Among all the methods, DCN-V2 delivered the best performance while remaining relatively efficient; DCN-Mix further reduced the cost, achieving a better trade-off between model efficiency and quality.

Can Cross Layers Replace ReLU layers? ReLU layers are the backbone for various Neural Nets including DNN, RNN [18, 32, 40] and CNN [23, 24, 42]. Our experiments in Tab. 4 showed encouraging results that CrossNet consistently outperformed ReLU-based DNN with the same model capacity. The memory was controlled by varying the depth, width and rank ({128, 256}) of layers. This is a very interesting preliminary study and sheds light for our future explorations on cross layers and design of DNNs.

Table 3: LogLoss and AUC (test) on Criteo and Movielen-1M. The metrics were averaged over 5 independent runs with their stddev in the parenthesis. In the 'Best Setting' column, the left reports DNN setting and the right reports model-specific setting. l denotes layer depth; n denotes CIN layer size; h and e, respectively, denotes #heads and att-embed-size; K denotes #experts and r denotes total rank.

Baseline	Criteo						MovieLens-1M			
	Logloss	AUC	Params	FLOPS	Best Setting		Logloss	AUC	Params	FLOPS
PNN	0.4421 (5.8E-4)	0.8099 (6.1E-4)	3.1M	6.1M	(3, 1024)	OPNN	0.3182 (1.4E-3)	0.8955 (3.3E-4)	54K	110K
DeepFm	0.4420 (1.4E-4)	0.8099 (1.5E-4)	1.4M	2.8M	(2, 768)	–	0.3202 (1.0E-3)	0.8932 (7.7E-4)	46K	93K
DLRM	0.4427 (3.1E-4)	0.8092 (3.1E-4)	1.1M	2.2M	(2, 768)	[512,256,64]	0.3245 (1.1E-3)	0.8890 (1.1E-3)	7.7K	16K
xDeepFm	0.4421 (1.6E-4)	0.8099 (1.8E-4)	3.7M	32M	(3, 1024)	l=2, n=100	0.3251 (4.3E-3)	0.8923 (8.6E-4)	160K	990K
AutoInt+	0.4420 (5.7E-5)	0.8101 (2.6E-5)	4.2M	8.7M	(4, 1024)	l=2, h=2, e=40	0.3204 (4.4E-4)	0.8928 (3.9E-4)	260K	500K
DCN	0.4420 (1.6E-4)	0.8099 (1.7E-4)	2.1M	4.2M	(2, 1024)	l=4	0.3197 (1.9E-4)	0.8935 (2.1E-4)	110K	220K
DNN	0.4421 (6.5E-5)	0.8098 (5.9E-5)	3.2M	6.3M	(3, 1024)	–	0.3201 (4.1E-4)	0.8929 (2.3E-4)	46K	92K
Ours										
DCN-V2	**0.4406 (6.2E-5)**	**0.8115 (7.1E-5)**	3.5M	7.0M	(2, 768)	l=2	0.3170 (3.6E-4)	0.8950 (2.7E-4)	110K	220K
DCN-Mix	0.4408 (1.0E-4)	0.8112 (9.8E-5)	2.4M	4.8M	(2, 512)	l=3, K=4, r=258	**0.3160 (4.9E-4)**	**0.8964 (2.9E-4)**	110K	210K
CrossNet	0.4413 (2.5E-4)	0.8107 (2.4E-4)	2.1M	4.2M	–	l=4, K=4, r=258	0.3185 (3.0E-4)	0.8937 (2.7E-4)	65K	130K

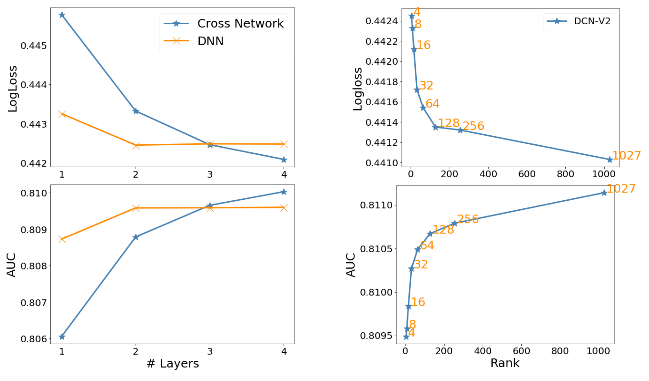
Table 4: Logloss and AUC (test) with a fixed memory budget.

#Params		7.9E+05	1.3E+06	2.1E+06	2.6E+06
LogLoss	CrossNet	**0.4424**	**0.4417**	**0.4416**	**0.4415**
	DNN	0.4427	0.4426	0.4423	0.4423
AUC	CrossNet	**0.8096**	**0.8104**	**0.8105**	**0.8106**
	DNN	0.8091	0.8094	0.8096	0.80961

7.3 How the Choice of Hyper-parameters Affect DCN-V2 Model Performance (RQ3)

This section examines the model performance as a function of hyper-parameters that include 1) depth of cross layers; 2) matrix rank of DCN-Mix; 3) number of experts in DCN-Mix.

Depth of Cross Layer. By design, the highest feature cross order captured by the cross net increases with layer depth. Hence, we constrain ourselves to the full-rank cross layers, and evaluate the performance change with layer depth.



(a) Layer depth **(b) Matrix rank**

Figure 5: Logloss and AUC (test) v.s. depth & matrix rank.

Fig. 5a shows the test LogLoss and AUC while increasing layer depth on the Criteo dataset. We see a steady quality improvement with a deeper cross network, indicating that it's able to capture more meaningful crosses. The rate of improvement, however, slowed down when more layers were used. This suggests the contribution from that of higher-order crosses is less significant than those from lower-order crosses. We also used a same-sized DNN as a reference. When there were ≤ 2 layers, DNN outperformed the cross network; when more layers became available, the cross network started to close the performance gap and even outperformed DNN. In the small-layer regime, the cross network could only approximate very low-order crosses (*e.g.,* $1 \sim 2$); in the large-layer regime, those low-order crosses were characterized with more parameters, and those high-order interactions were started to be captured.

Rank of Matrix. The rank of the weight matrix controls the # of params and the portion of low-frequency signals passing through the cross layers. Hence, we study its influence on model quality. The model based on a well-performed setting with 3 cross layers followed by 3 512-size ReLU layers. We approximate the matrix W in each cross layer by $UV^\top$ where $U, V \in \mathbb{R}^{d \times r}$, and vary r.

Fig. 5b shows the test LogLoss v.s. matrix's rank r on Criteo. When r was as small as 4, the performance was on-par with other baselines. When r was increased from 4 to 64, the LogLoss decreased almost linearly with r (*i.e.,* model's improving). When r was further increased from 64 to full, the improvement on LogLoss slowed down. We refer to 64 as the *threshold rank*. The significant slow down from 64 suggests that the important signals characterizing feature crosses could be captured in the top-64 singular values.

Our hypothesis for the value of this *threshold rank* is $O(k)$ where k represents # features (39 for Criteo). Consider the (i, j)-th block of matrix W, we can view $W_{i,j} = W_{i,j}^L + W_{i,j}^H$, where $W_{i,j}^L$ stores the dominant signal (low-frequency) and $W_{i,j}^H$ stores the rest (high-frequency). In the simplest case where $W_{i,j}^L = c_{ij}\mathbf{1}\mathbf{1}^\top$, the entire matrix W^L will be of rank k. The effectiveness of this hypothesis remains to be verified across multiple datasets.

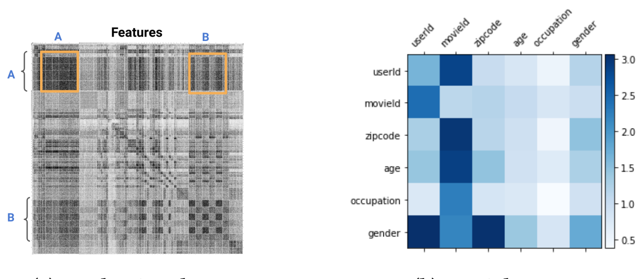
Number of Experts. We've observed that 1) best-performed setting (#expert, gate, matrix activation type) was subjective to datasets and model architectures; 2) the best-performed model of each setting yielded similar results. For example, for a 2-layered cross net with total rank 256 on Criteo, the LogLoss for 1, 4, 8, 16, and 32 experts, respectively, was 0.4418, 0.4416, 0.4416, 0.4422, and 0.4420. The fact that more lower-rank experts weren't performing better than a single higher-rank expert might be caused by the naïve gating functions and optimizations adopted. We believe more sophisticated gating [21, 28, 29] and optimizations (*e.g.*, alternative training, special initialization, temperature adjustment) would yield better results with a mixture of experts architecture. This, however, is beyond the scope of this paper and we leave it to future work.

7.4 Model Understanding (RQ4)

A good understanding of the learned feature crosses is crucial to fields like ML fairness and ML for health. Fortunately, the weight matrix W in DCN-V2 exactly reveals what feature crosses the model has learned to be important. Assuming the i-th feature is embedded as $\mathbf{x}_i$, then, the block-wise view shows that the importance of interaction between $\mathbf{x}_i$ and $\mathbf{x}_j$ is characterized by block $W_{i,j}$:

$$\mathbf{x} \odot W\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_k \end{bmatrix} \odot \begin{bmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,k} \\ W_{2,1} & W_{2,2} & \cdots & W_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ W_{k,1} & W_{k,2} & \cdots & W_{k,k} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_k \end{bmatrix}$$

Fig. 6 shows the learned weight matrix W in the first cross layer. Subplot (a) shows the entire matrix with orange boxes highlighting some notable feature crosses. The off-diagonal block corresponds to crosses that are known to be important, suggesting the effectiveness of DCN-V2. The diagonal block represents self-interaction (x^2's). Subplot (b) indicates some strong interactions learned, *e.g.*, `Gender` × `UserId`, `MovieId` × `UserId`.



(a) Production data **(b) Movielen-1M**

Figure 6: Learned weight matrix in DCN-V2.
Rows/cols represent features. For (a), feature names were omitted for proprietary reasons; darker pixel represents larger (abs) weight. For (b), each block represents its Frobenius norm.

8 Productionizing DCN-V2 at Google

This section provides a case study to share our experience productionizing DCN-V2 in a large-scale recommender system in Google. We've achieved significant gains through DCN-V2 in both offline model accuracy, and online key business metrics. The gains are also more pronounced than public datasets, which might be due to the significantly larger data size and more complex data distributions in production datasets.

Production Data and Model: The production data are sampled user logs consisting of hundreds of billions of training examples. The vocabulary sizes of sparse features vary from 2 to millions. The baseline model is a fully-connected multi-layer perceptron (MLP) with ReLU activations.

Comparisons with Production Models: When compared with production model, DCN-V2 yielded 0.6% AUCLoss (*i.e.*, 1 - AUC) improvement. For this particular model, a gain of 0.1% on AUCLoss is considered significant. We also observed significant online key business metrics gains. Tab. 5 further verifies the amount of gain from cross layers by replacing cross layers with same-sized ReLU layers.

Table 5: Relative AUCLoss of DCN-V2 v.s. same-sized ReLUs

1layer ReLU	2layer ReLU	1layer DCN-V2	2layer DCN-V2
0%	-0.15%	-0.19%	-0.45%

Practical Learnings. We share some practical lessons we have learned through productionizing DCN-V2.

- It's better to insert the cross layers in between the input and the hidden layers of DNN (also observed in [44]). Our hypothesis is that the physical meaning of feature representations and their interactions becomes weaker as it goes farther away from the input layer.
- We saw consistent accuracy gains by stacking or concatenating 1 - 2 cross layers. Beyond 2 cross layers, the gains start to plateau.
- We observed that both stacking cross layers and concatenating cross layers work well. Stacking layers learns higher-order feature interactions, while concatenating layers (similar to multi-head mechanism [48]) captures complimentary interactions.
- We observed that using low-rank DCN with rank (input size)/4 consistently preserved the accuracy of a full-rank DCN-V2.

9 Conclusions and Future Work

In this paper, we propose a new model—DCN-V2—to model explicit crosses in an expressive yet simple manner. Observing the low-rank nature of the weight matrix in the cross network, we also propose a mixture of low-rank DCN (DCN-Mix) to achieve a healthier trade-off between model performance and latency. DCN-V2 has been successfully deployed in multiple web-scale learning to rank systems with significant offline model accuracy and online business metric gains. Our experimental results also have demonstrated DCN-V2's effectiveness over SOTA methods.

For future work, we are interested in advancing our understanding of 1). the interactions between DCN-V2 and optimization algorithms such as second-order methods; 2). the relation between embedding, DCN-V2 and its rank of matrix. Further, we would like to improve the gating mechanism in DCN-Mix. Moreover, observing that cross layers in DCN-V2 may serve as potential alternatives to ReLU layers in DNNs, we are very interested to verify this observation across more complex model architectures (*e.g.*, RNN, CNN).

Acknowledgments

We would like to thank Bin Fu, Gang (Thomas) Fu, and Mingliang Wang for their early contributions of DCN-V2; Anusha Sriraman, Eu-Jin Goh, Evan Ettinger, Jakob Bauer, Jason Trader, Jie (Jerry) Zhang, Julian Grady, Kevin Regan, Samuel Ieong, Shuying Zhang, Sugato Basu, Tianshuo Deng, Todd Phillips, Wenjing Ma, Yang Li, Yayang Tian, Zhe (Shirley) Chen and many others for their efforts and supports in productionizing DCN-V2 (in alphabetical order); Ting Chen for his initial idea of mixture of low-rank; and Maciej Kula for his help in open-sourcing the code and tutorial in TensorFlow Recommenders (TFRS).

References

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

[2] 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.

[3] Andrei Z Broder. 2008. Computational advertising and recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*. 1–2.

[4] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. 129–136.

[5] Ting Chen, Ji Lin, Tian Lin, Song Han, Chong Wang, and Denny Zhou. 2018. Adaptive mixture of low-rank factorizations for compact neural modeling. (2018).

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & Deep Learning for Recommender Systems. *arXiv preprint arXiv:1606.07792* (2016).

[7] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2019. Adaptive Factorization Network: Learning Adaptive-Order Feature Interactions. *arXiv preprint arXiv:1909.03276* (2019).

[8] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 101–109.

[9] Petros Drineas and Michael W Mahoney. 2005. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *journal of machine learning research* 6, Dec (2005), 2153–2175.

[10] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. 2013. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314* (2013).

[11] Yuwei Fan, Jordi Feliu-Faba, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. 2019. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences* 6, 2 (2019), 21.

[12] Gene H Golub and Charles F Van Loan. 1996. Matrix Computations Johns Hopkins University Press. *Baltimore and London* (1996).

[13] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[14] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[16] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 355–364.

[17] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 5–53.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[19] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.

[20] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866* (2014).

[21] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[22] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks* 8, 1 (1997), 98–113.

[24] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.

[25] Zeyu Li, Wei Cheng, Yang Chen, Haifeng Chen, and Wei Wang. 2020. Interpretable Click-Through Rate Prediction through Hierarchical Attention. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 313–321.

[26] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1754–1763.

[27] Tie-Yan Liu. 2011. *Learning to rank for information retrieval*. Springer Science & Business Media.

[28] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. *arXiv preprint arXiv:1712.01312* (2017).

[29] Jiaqi Ma, Zhe Zhao, Jilin Chen, Ang Li, Lichan Hong, and Ed H Chi. 2019. Snr: Sub-network routing for flexible parameter sharing in multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 216–223.

[30] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1930–1939.

[31] Hrushikesh N Mhaskar. 1996. Neural networks for optimal approximation of smooth and analytic functions. *Neural computation* 8, 1 (1996), 164–177.

[32] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 5528–5531.

[33] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. 2020. A metric learning reality check. *arXiv preprint arXiv:2003.08505* (2020).

[34] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[35] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1149–1154.

[36] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[37] Steffen Rendle. 2012. Factorization Machines with libFM. *ACM Trans. Intell. Syst. Technol.* 3, 3, Article 57 (May 2012), 22 pages.

[38] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. *arXiv preprint arXiv:2005.09683* (2020).

[39] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.

[40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[41] J Ben Schafer, Joseph Konstan, and John Riedl. 1999. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*. 158–166.

[42] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[43] Frank Seide, Gang Li, Xie Chen, and Dong Yu. 2011. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 24–29.

[44] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 255–262.

[45] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[46] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1161–1170.

[47] Gregory Valiant. 2014. Learning polynomials with neural networks. (2014).

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[49] Andreas Veit, Michael J Wilber, and Serge Belongie. 2016. Residual Networks Behave Like Ensembles of Relatively Shallow Networks. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 550–558.

[50] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17*. 1–7.

[51] Ruoxi Wang, Yingzhou Li, Michael W Mahoney, and Eric Darve. 2019. Block Basis Factorization for Scalable Kernel Evaluation. *SIAM J. Matrix Anal. Appl.* 40, 4 (2019), 1497–1526.

[52] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617* (2017).

[53] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7370–7379.

Appendix

10 Baseline performance reported in papers

Tab. 6a lists the quoted Logloss and AUC metrics reported in papers for each baseline.

11 Theorem Proofs

11.1 Proofs for Bitwise Interactions.

Proof. We start with notations; then prove by induction.

Notations. Let $[k] := \{1, \ldots, k\}$. Let's denote the embedding as $\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_c]$, the output from the l-th cross layer to be $\mathbf{x}^l = [\mathbf{x}_1^l; \mathbf{x}_2^l; \ldots; \mathbf{x}_c^l]$ where $\mathbf{x}_i, \mathbf{x}_i^l \in \mathbb{R}^{e_i}$ and e_i is the embedding size for the i-th feature. To simplify the notations, let's also define the feature interaction between features in an ordered set I (*e.g.,* (i_1, i_3, i_4)) with weights characterized by an ordered set J as

$$g(I, J; \mathbf{x}, W) = \mathbf{x}_{i_1} \odot \left(W_{i_1, i_2}^{j_1} \mathbf{x}_{i_2} \odot \ldots \odot \left(W_{i_k, i_{k+1}}^{j_k} \mathbf{x}_{i_{l+1}}\right)\right) \tag{6}$$

where weights W_{i_a, i_b}^{j} represents the (i_a, i_b)-th block in weight W^j at the j-th cross layer, and it serves as two purposes: align the dimensions between features and increase the impressiveness of the feature cross representations. Note that given the order of $\mathbf{x}_i$'s, the subscripts of matrix W's are uniquely determined.

Proposition. We first proof by induction that $\mathbf{x}_i^l$ has the following formula:

$$\mathbf{x}_i^l = \sum_{p=2}^{l+1} \sum_{I \in S_p^i} \sum_{J \in C_l^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_i \tag{7}$$

where S_p^i is a set which represents all the combinations of choosing p elements from $[c]$ with replacement, and with first element fixed to be i: $S_p^i =: \left\{\mathbf{y} \in [c]^p \;\middle|\; y_1 = i\right\}, \ \forall I \in S_p, \ I = (i_1, \ldots, i_p)$; and C_l^{p-1} is a set that represents choosing a combination of $p-1$ indices out of integers $[l]$ at a time: $C_l^{p-1} := \left\{\mathbf{y} \in [l]^{p-1} \;\middle|\; \forall i < j, y_i > y_j\right\}$.

Base case. When $l = 1$, $\mathbf{x}_i^1 = \sum_j W_{i,j}^1 \mathbf{x}_j + \mathbf{x}_i$.

Induction step. Let's assume that when $l = k$,

$$\mathbf{x}_i^k = \sum_{p=2}^{k+1} \sum_{I \in S_p^i} \sum_{J \in C_k^{p-1}} g_J(\mathbf{x}; I) + \mathbf{x}_i$$

Then, for $l = k + 1$, we have

$$\begin{aligned}
\mathbf{x}_i^{k+1} &= \mathbf{x}_i \odot \sum_{q=1}^{c} W_{i,q}^{k+1} \mathbf{x}_q^k + \mathbf{x}_i^k \\
&= \mathbf{x}_i \odot \sum_{q=1}^{c} W_{i,q}^{k+1} \left(\sum_{p=2}^{k+1} \sum_{I \in S_p^q} \sum_{J \in C_k^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_q \right) + \\
&\quad \sum_{p=2}^{k+1} \sum_{I \in S_p^i} \sum_{J \in C_k^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_i \\
&= \sum_{q=1}^{c} \sum_{p=2}^{k+1} \sum_{I \in S_p^q} \sum_{J \in C_k^{p-1}} \mathbf{x}_i \odot \left(W_{i,q}^{k+1} g(I, J; \mathbf{x}, W) \right) + \\
&\quad \sum_{q=1}^{c} \mathbf{x}_i \odot W_{i,q}^{k+1} \mathbf{x}_q + \sum_{p=2}^{k+1} \sum_{I \in S_p^i} \sum_{J \in C_k^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_i \\
&= \sum_{p=2}^{k+1} \sum_{J \in C_k^{p-1}} \sum_{q=1}^{c} \sum_{I \in S_p^q} \mathbf{x}_i \odot \left(W_{i,q}^{k+1} g(I, J; \mathbf{x}, W) \right) + \\
&\quad \sum_{p=2} \sum_{J=k+1} \sum_{I \in S_2^i} g(I, J; \mathbf{x}, W) + \sum_{p=2}^{k+1} \sum_{I \in S_p^i} \sum_{J \in C_k^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_i \\
&= \sum_{p=2}^{k+1} \sum_{J \in k+1 \oplus C_k^{p-1}} \sum_{I \in S_{p+1}^i} g(I, J; \mathbf{x}, W) + \\
&\quad \sum_{p=2} \sum_{J=k+1} \sum_{I \in S_2^i} g(I, J; \mathbf{x}, W) + \sum_{p=2}^{k+1} \sum_{I \in S_p^i} \sum_{J \in C_k^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_i \\
&= \left(\sum_{p=3}^{k+2} \sum_{J \in k+1 \oplus C_k^{p-2}} \sum_{I \in S_p^i} + \sum_{p=3}^{k+1} \sum_{I \in S_p^i} \sum_{J \in C_k^{p-1}} \right) g(I, J; \mathbf{x}, W) + \\
&\quad \left(\sum_{p=2} \sum_{I \in S_2^i} \sum_{J \in C_k^1} g(I, J; \mathbf{x}, W) + \sum_{p=2} \sum_{J=k+1} \sum_{I \in S_2^i} \right) g(I, J; \mathbf{x}, W) + \mathbf{x}_i \\
&= \sum_{p=3}^{k+2} \sum_{J \in C_{k+1}^{p-1}} \sum_{I \in S_p^i} g(I, J; \mathbf{x}, W) + \sum_{p=2} \sum_{J = C_{k+1}^{p-1}} \sum_{I \in S_p^i} g(I, J; \mathbf{x}, W) + \mathbf{x}_i \\
&= \sum_{p=2}^{k+2} \sum_{I \in S_p^i} \sum_{J \in C_{k+1}^{p-1}} g(I, J; \mathbf{x}, W) + \mathbf{x}_i
\end{aligned}$$

where $\oplus$ denotes adding index $k+1$ to each element in the set of C_k^{p-1}. The first 5 equalities are are straightforward. For the 6th equality, we first interchanged variable $p' = p + 1$ for the first term, and separated the third term into cases of $p = 2$ and $p > 2$. Then, we group the terms into two cases: $p = 2$ and $p > 2$. For the second to the last equality, we combined the summations over J. Consider the set of choosing a combination of $p - 1$ indices from $k + 1$ integers, it could be separated into two sets, with index $k + 1$ and without. Hence, $C_{k+1}^{p-1} = C_k^{p-1} \cup \left((k+1) \oplus C_k^{p-2}\right)$.

Conclusion. Since both the base case and the induction step hold, we conclude that $\forall\, l \geq 1$, Eq (7) holds. This completes the proof.

Table 6: Baseline performance reported in papers and their complexities.

(a) Baseline performance reported in papers. The metrics (Logloss, AUC) are quoted from papers. Each row represents a baseline, each column represents the paper where the metrics are being reported. The best metric for each baseline is marked in bold.

Model \ Paper	DeepFM[13] (2017)	DCN[50] (2017)	xDeepFM[26] (2018)	DLRM[34] (2019)	AutoInt[46] (2019)	DCN-V2 (ours)
DeepFM	(0.45083, 0.8007)	–	(0.4468, 0.8025)	–	(0.4449, 0.8066)	**(0.4420, 0.8099)**
DCN	–	(0.4419, -)	(0.4467, 0.8026)	(-, ∼ 0.789)	(0.4447, 0.8067)	**(0.4420, 0.8099)**
xDeepFM	–	–	(**0.4418**, 0.8052)	–	(0.4447, 0.8070)	(0.4421, **0.8099**)
DLRM	–	–	–	(-, ∼ 0.790)	–	(0.4427, **0.8092**)
AutoInt	–	–	–	–	(0.4434, 0.8083)	**(0.4420, 0.8101)**
DCN-V2	–	–	–	–	–	**(0.4406, 0.8115)**
DNN	–	(0.4428, -)	(0.4491, 0.7993)	–	–	**(0.4421, 0.8098)**

(b) Complexity table for DCN v2 and baselines. Some common notations are: d denotes the input dimension; e denotes embedding dimension for each feature; N_c and N_d, respectively, denotes the number of categorical features and dense feature; L denotes the hidden layer depth and h denotes the hidden layer size. For DLRM, h_b, L_b, respectively, denotes bottom MLP layer size and depth (for simplicity we assume equal-sized bottom layers.); For DCN and DCN v2, L_c denotes cross network depth, r denotes the total rank; For xDeepFM, h_c and L_c, respectively, denotes CIN layer size and depth. For AutoInt; L_a, e_a, and H_a, denotes attention layer depth, attention embedding size and number of attention heads, respectively.

Model	#Parameters		FLOPS
	Interaction	DNN	
DNN	NA	$O(dh + h^2(L-1))$	$O(2\#\text{params})$
PNN (OPNN) [35]	$O(e^2 N_c(N_c-1)/2)$	$O((d + N_c(N_c-1)/2)h + h^2(L-1))$	$O(2\#\text{params})$
DeepFM [13]	$O(d)$	$O(dh + h^2(L-1))$	$O(2\#\text{params} + eN_c(N_c-1))$
DLRM [34]	$O(N_d h_b + h_b^2(L_b-1) + eh_b)$	$O((N_c+1)N_c/2)h + h^2(L-1)$	$O(e(N_c+1)N_c + 2\#\text{params})$
DCN [50]	$O(dL_c)$	$O(dh + h^2(L-1))$	$O(2\#\text{params})$
xDeepFM [26]	$O(L_c h_c(1+h_c)N_c)$	$O(dh + h^2(L-1))$	$O(2eN_c^2 h_c + 2eN_c h_c^2(L_c-1) + 2(dh + h^2(L-1))$
AutoInt [46]	$O(4L_a e_a e H_a)$	$O(dh + h^2(L-1))$	$O(4(e_a e N_c + N_c^2 e_a) + 2(dh + h^2(L-1)))$
DCN-V2 (ours)	$O(d^2 L_c)$ $O(2drL_c)$ (for low-rank version)	$O(dh + h^2(L-1))$	$O(2\#\text{params})$

In such case, the l-th cross layer contains all the feature interactions (feature-wise) of order up to $l+1$. The interactions between different feature set is parameterized differently, specifically, the interactions between features in set I (feature's can be repeated) of order p is

$$\sum_{\mathbf{i}\in I'} \sum_{\mathbf{j}\in C_p^{p-1}} \left\{ g(\mathbf{i},\mathbf{j};\mathbf{x},W) = \mathbf{x}_{i_1} \odot \left(W_{i_1,i_2}^{j_1} \mathbf{x}_{i_2} \odot \ldots \odot \left(W_{i_k,i_{k+1}}^{j_k} \mathbf{x}_{i_{l+1}} \right)\right)\right\}$$

where I' contains all the permutations of elements in I.

□

11.2 Proofs for Feature-wise interactions.

Proof. Instead of treating each feature embedding as a unit, we treat each element x_i in input embedding $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ as a unit. This is a special case of subsection 11.1 where all the feature embedding sizes are 1. In such case, all the computations are interchangeable. Hence, we adopt the notations and also the result of Equation 7, that is, the i-th element in the l-th layer of cross network $\mathbf{x}^l$ has the following formula:

$$x_i^l = \sum_{p=2}^{l+1} \sum_{I\in S_p^i} \sum_{J\in C_l^{p-1}} g(I,J;\mathbf{x},W) + x_i \tag{8}$$

To ease the proof and simplify the final formula, we assume the final logit for a l-layer cross network is $\mathbf{1}^\top \mathbf{x}^l$, then

$$\begin{aligned}
\mathbf{1}^\top \mathbf{x}^l &= \sum_{i=1}^{d} \sum_{p=2}^{l+1} \sum_{I\in S_p^i} \sum_{J\in C_l^{p-1}} x_{i_1} \odot \left(w_{i_1 i_2}^{(j_1)} x_{i_2} \odot \ldots \odot \left(w_{i_k i_{k+1}}^{(j_k)} x_{i_{l+1}} \right)\right) + \sum_{i=1}^{d} x_i \\
&= \sum_{p=2}^{l+1} \sum_{I\in S_p} \sum_{J\in C_l^{p-1}} w_{i_1 i_2}^{(j_1)} \ldots w_{i_k i_{k+1}}^{(j_k)} x_{i_1} x_{i_2} \ldots x_{i_{l+1}} + \sum_{i=1}^{d} x_i \\
&= \sum_{p=2}^{l+1} \sum_{|\boldsymbol{\alpha}|=p} \sum_{J\in C_l^{p-1}} \sum_{\mathbf{i}\in P_{\boldsymbol{\alpha}}} \prod_{k=1}^{|\boldsymbol{\alpha}|-1} w_{i_k i_{k+1}}^{(j_k)} x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d} + \sum_{i=1}^{d} x_i \\
&= \sum_{\boldsymbol{\alpha}} \sum_{\mathbf{j}\in C_l^{|\boldsymbol{\alpha}|-1}} \sum_{\mathbf{i}\in P_{\boldsymbol{\alpha}}} \prod_{k=1}^{|\boldsymbol{\alpha}|-1} w_{i_k i_{k+1}}^{(j_k)} x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d} + \sum_{i=1}^{d} x_i
\end{aligned}$$

where $P_{\boldsymbol{\alpha}}$ is the set of all the permutations of $(\underbrace{1\cdots1}_{\alpha_1 \text{ times}} \cdots \underbrace{d\cdots d}_{\alpha_d \text{ times}})$, $C_l^{|\boldsymbol{\alpha}|-1}$ is a set that represents choosing a combination of $|\boldsymbol{\alpha}|-1$ indices out of integers $\{1,\cdots,l\}$ at a time, specifically,

$$C_l^{|\boldsymbol{\alpha}|-1} := \left\{ \mathbf{y} \in [l]^{|\boldsymbol{\alpha}|-1} \,\middle|\, (y_i \neq y_j) \wedge (y_{j_1} > y_{j_2} > \ldots > y_{j_{|\boldsymbol{\alpha}|-1}}) \right\}.$$

The second equality combined the first and the third summations into a single one summing over a new set $S_p^c := [c]^p$. The third equality re-represented the cross terms (monomials) using multi-index $\boldsymbol{\alpha}$, and modified the index for weights w's accordingly. The last equality combined the first two summations. Thus the proof. $\square$

12 Complexity Table

The complexity (#params and FLOPS) for our method and for the baselines can be found at Table **??**.