Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features

Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu*, JC Mao
Bing Ads, Microsoft Research*
Microsoft Corporation
One Microsoft Way, Redmond, WA

ABSTRACT

Manually crafted combinatorial features have been the "secret sauce" behind many successful models. For web-scale applications, however, the variety and volume of features make these manually crafted features expensive to create, maintain, and deploy. This paper proposes the *Deep Crossing* model which is a deep neural network that automatically combines features to produce superior models. The input of Deep Crossing is a set of individual features that can be either dense or sparse. The important crossing features are discovered implicitly by the networks, which are comprised of an embedding and stacking layer, as well as a cascade of *Residual Units*.

Deep Crossing is implemented with a modeling tool called the Computational Network Tool Kit (CNTK), powered by a multi-GPU platform. It was able to build, from scratch, two web-scale models for a major paid search engine, and achieve superior results with only a sub-set of the features used in the production models. This demonstrates the potential of using Deep Crossing as a general modeling paradigm to improve existing products, as well as to speed up the development of new models with a fraction of the investment in feature engineering and acquisition of deep domain knowledge.

Keywords

Neural Networks, Deep Learning, Convolutional Neural Network (CNN), Deep Crossing, Combinatorial Features, DSSM, Residual Net, CNTK, GPU, Multi-GPU Platform

1. INTRODUCTION

Traditional machine learning algorithms are supposed to make the best use of all input features to predict and classify new instances. However simply using raw features rarely provides optimal results. Therefore in both industry and academia, there exists a large body of work on engineering the transformation of raw features. One major type of transformation is to construct functions based on a combination of multiple features, and use their output as the input to a learner. These *combinatorial features*[1] are sometimes called *cross features*, or *multi-way features*.

Combinatorial features are powerful tools, especially in the hands of domain experts. In our own experiences with a major sponsored search engine, they are among the strongest features in many models. In the Kaggle community, top data scientists are the masters of crafting such features, even crossing three to five dimensions[2]. The intuition and the ability to create effective combinatorial features is an essential ingredient in their winning formulas. In the computer vision community, SIFT-like [13] features were the key drivers behind the then state-of-the-art performance of ImageNet competitions. SIFT features are extracted on image patches and are special forms of combinatorial features.

The power of combinatorial features comes with high cost. There is a steep learning curve to climb before an individual can start to create meaningful features. As the number of features grows, managing, maintaining, and deploying them becomes challenging. In web-scale applications, finding additional combinatorial features to improve an existing model is a daunting task due to the large search space, and the slow turnaround in the training and evaluation cycle given billions of samples.

Deep Learning [9, 20, 16] carries the promise of learning from individual features without manual intervention [17, 21]. Speech [8, 4, 18] and image recognition [11] were among the first to demonstrate this potential. Convolution kernels learned from specific tasks through deep Convolutional Neural Networks (CNN) have replaced the hand-crafted SIFT-like features as the state-of-the-art in image recognition. A similar model [3] has been applied to NLP applications to build language processing models from scratch without extensive feature engineering.

Deep Crossing extends the success of deep learning to a more general setting where individual features are of a different nature. More specifically, it takes in individual features such as text, categorical, ID, and numerical features, and searches for the optimal combinations automatically based on the specific task. Furthermore, Deep Crossing is designed to handle web-scale applications and data sizes. This is not only because the authors are primarily interested in such applications, but also because there are not many options for general purpose models operating at this scale. It is

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939704

[1]See Sec. 4 for a formal definition and concrete examples.
[2]See github.com/owenzhang/Kaggle-AmazonChallenge2013 for an example of multi-way features in Owen Zhang's winning solution for the Amazon's Employee Access Challenge.

Feature name	Type	Dimension
Query	Text	49,292
Keyword	Text	49,292
Title	Text	49,292
MatchType	Category	4
CampaignID	ID	10,001
CampaignIDCount	Numerical	5

Table 1: Examples of individual features

worth noting that Deep Crossing does in some way generate combinatorial features during the learning process, although the output of Deep Crossing is a model that does not have explicit representations for these features.

2. RELATED WORK

The idea of learning deep neural network without manually crafted features is not new. In early 80s, Fukushima [6] reported a seven-layer Neocognitron network that recognized digits from raw pixels of images. By utilizing a partially connected structure, Neocognitron achieved shift invariance which is an important property for visual recognition tasks. CNN was invented by LeCun et. al. [12] in the late 90s with a similar architecture, especially the partially connected convolution kernels. Despite the solid foundation of CNN as a recognition engine, classifiers based on SIFT-like [13] features dominated image recognition for nearly a decade. In 2012, Krizhevsky et. al. proposed the AlexNet [11] that beat the SIFT-based baseline error rate by almost 11 absolute percentage points. Recently, a 152-layer Residual Net [7] won both the ImageNet and the MS COCO Competitions in 2015.

The evolution of deep CNN is both inspiring and encouraging. It demonstrated that deep learning is able to improve even in systems in which the best manual features have been finely tuned over a decade. In other words, even the most experienced domain experts could miss the deep interactions among features that were captured by deep CNNs using task-specific filters. Realizing this has profound implications on our work on Deep Crossing.

Deep Semantic Similarity Model (DSSM) [10] learns the semantic similarity between a pair of text strings, each represented by a sparse representation called tri-letter grams. The learning algorithm optimizes an objective function based on cosine distance by embedding the tri-letter grams into two vectors. The learned embedding captures the semantic meaning of words and sentences, and has been applied to sponsored search, question answering, and machine translation with strong results.

Factorization Machines (FM) [15] in their general form models d-way interactions among individual features. In the presence of very sparse inputs, FM shows better results than SVMs but it's unclear how it performs on dense features.

For NLP tasks, [3] built a unified neural network architecture that avoids task-specific feature engineering. Deep Crossing aims at a broader range of input features.

3. SPONSORED SEARCH

Deep Crossing is discussed in the context of *sponsored search* of a major search engine. Readers can refer to [5] for an overview on this subject. In brief, sponsored search is responsible for showing ads alongside organic search results. There are three major agents in the ecosystem: the user, the advertiser, and the search platform. The goal of the platform is to show the user the advertisement that best matches the user's intent, which was expressed mainly through a specific query. Below are the concepts key to the discussion that follows.

Query: A text string a user types into the search box

Keyword: A text string related to a product, specified by an advertiser to match a user query

Title: The title of a sponsored advertisement (referred to as "an ad" hereafter), specified by an advertiser to capture a user's attention

Landing page: A product's web site a user reaches when the corresponding ad is clicked by a user

Match type: An option given to the advertiser on how closely the keyword should be matched by a user query, usually one of four kinds: *exact, phrase, broad* and *contextual*

Campaign: A set of ads that share the same settings such as budget and location targeting, often used to organize products into categories

Impression: An instance of an ad being displayed to a user. An impression is usually logged with other information available at run-time

Click: An indication of whether an impression was clicked by a user. A click is usually logged with other information available at the run-time

Click through rate: Total number of clicks over total number of impressions

Click Prediction: A critical model of the platform that predicts the likelihood a user clicks on a given ad for a given query

Sponsored search is only one kind of web-scale application. However, given the richness of the problem space, the various types of features, and the sheer volume of data, we think our results can be generalized to other applications with similar scale.

4. FEATURE REPRESENTATION

This section defines and compares combinatorial features and individual features, using examples listed in Table 1. The features in the table are available during run-time when an ad is displayed (an impression). They are also available in offline logs for model training, etc.

4.1 Individual Features

Each individual feature X_i is represented as a vector. For text features such as a query, one option is to convert the string into a tri-letter gram with $49,292$ dimensions as in [10]. Categorical input such as `MatchType` is represented by a one-hot vector, where exact match (see Sec. 3) is $[1,0,0,0]$, phrase match is $[0,1,0,0]$, and so on.

There are usually millions of campaigns in a sponsored search system. Simply converting campaign ids into a one-hot vector would significantly increase the size of the model (see Sec. 5.1). One solution is to use a pair of companion features as exemplified in the table, where `CampaignID` is

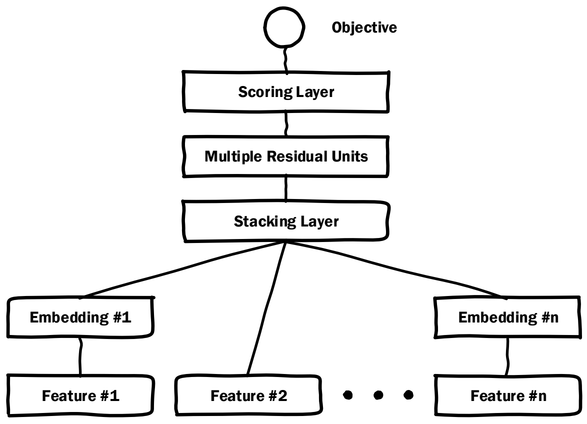


Figure 1: Deep Crossing Model Architecture

a one-hot representation consisting only of the top 10,000 campaigns with the highest number of clicks. The 10,000th slot (index starts from 0) is saved for all the remaining campaigns. Other campaigns are covered by `CampaignIDCount`, which is a numerical feature that stores per campaign statistics such as *click through rate*. Such features will be referred as a *counting feature* in the following discussions. All the features introduced so far are sparse features except the counting features.

4.2 Combinatorial Features

Given individual features $X_i \in \mathbb{R}^{n_i}$ and $X_j \in \mathbb{R}^{n_j}$, a *combinatorial feature* $X_{i,j}$ is defined in $\mathbb{R}^{n_i} \times \mathbb{R}^{n_j}$. Combinatorial features also have sparse and dense representations. An example of sparse representation is a `CampaignId`×`MatchType` feature, which is a one-hot vector of $10,001 \times 4 = 40,004$ dimensions. An example of a dense representation counts how many ad clicks for a specific `CampaignId` and `MatchType` combination. The dimension for the dense representation is the same as its sparse counterpart.

Deep Crossing avoids using combinatorial features. It works with both sparse and dense individual features, and supports a broad range of feature types described above. This gives users the freedom to use the features of their choice from their specific application. While collecting features and converting them into the right representations still requires a significant amount of effort, the work stops at the level of individual features. The rest is taken care of by the model.

5. MODEL ARCHITECTURE

Fig. 1 is the model architecture of Deep Crossing, where the input is a set of individual features. The model has four types of layers including the *Embedding*, the *Stacking*, the *Residual Unit*, and the *Scoring Layer*. The objective function is *log loss* for our applications but can be easily customized to soft-max or other functions. Log loss is defined as:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1-y_i)\log(1-p_i)), \quad (1)$$

where i indexes to the training samples, N is the number of samples, y_i is the per sample label, and p_i is the output of the one-node Scoring Layer in Fig. 1, which is a Sigmoid function in this case. The label y_i in the click prediction problem is a user click.

5.1 Embedding and Stacking Layers

Embedding is applied per individual feature to transform the input features. The embedding layer consists of a single layer of a neural network, with the general form:

$$X_j^O = \max(\mathbf{0}, \mathbf{W}_j X_j^I + \mathbf{b}_j), \quad (2)$$

where j indexes the individual feature, $X_j^I \in \mathbb{R}^{n_j}$ is the input feature, $\mathbf{W}_j$ is an $m_j \times n_j$ matrix, $\mathbf{b} \in \mathbb{R}^{n_j}$, and X_j^O is the embedded feature. When $m_j < n_j$, embedding is used to reduce the dimensionality of the input feature. The per element max operator is usually referred to as a *rectified linear unit* (ReLU) in the context of neural networks. The output features are then stacked (concatenated) into one vector as the input to the next layer:

$$X^O = [X_0^O, X_1^O, \cdots, X_K^O], \quad (3)$$

where K is the number of input features. Note that both $\{\mathbf{W}_j\}$ and $\{\mathbf{b}_j\}$ are the parameters of the network, and will be optimized together with the other parameters in the network. This is an important property of embedding in Deep Crossing. Unlike the embedding only approaches such as word2vec [14], it is an integral part of the overall optimization process.

It should be pointed out that the size of the embedding layers has significant impact on the overall size of the model. Even for sparse features, the $m_j \times n_j$ weight matrices are inherently dense. This is why Deep Crossing uses the companion features in Sec. 4 to constrain the dimensionality of high cardinality features. For the click prediction experiments in Sec. 7, the total size of the input features is around 200,000, and the m_j for the high dimensional features such as query and keyword are uniformly set to 256. Features with dimensionality lower than 256 are stacked without embedding. An example is `Feature #2` in Fig. 1.

5.2 Residual Layers

The residual layers are constructed from the *Residual Unit* in Fig. 2. The Residual Unit is the basic building block of the Residual Net [7] that claimed the world record in the ImageNet contest. Deep Crossing uses a slightly modified Residual Unit that doesn't use convolutional kernels. To our knowledge, this is the first time Residual Units have been used to solve problems beyond image recognition.

The unique property of Residual Unit is to add back the original input feature after passing it through two layers of ReLU transformations. Specifically:

$$X^O = \mathcal{F}(X^I, \{\mathbf{W}_0, \mathbf{W}_1\}, \{\mathbf{b}_0, \mathbf{b}_1\}) + X^I, \quad (4)$$

where $\mathbf{W}_{\{0,1\}}$ and $\mathbf{b}_{\{0,1\}}$ are the parameters of the two layers, and $\mathcal{F}$ denotes the function that maps the input X^I of the Residual Unit to the output X^O. Moving X^I to the left side of Eq. 4, $\mathcal{F}(\cdot)$ is essentially fitting the residual of $X^O - X^I$. In [7] the authors believed that fitting residuals has a numerical advantage. While the actual reason why Residual Net could go as deep as 152 layers with high performance is subject to more investigations, Deep Crossing did exhibit a few properties that might benefit from the Residual Units.

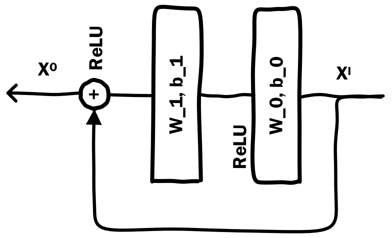


Figure 2: A Residual Unit

Before Deep Crossing, we tried a number of model architectures with deep layers but none of them provided significant gains over a model with two to three layers to justify the added complexity. Deep Crossing is our strongest model that easily beats the performance of its shallower counterparts.

Deep Crossing was applied to a wide variety of tasks. It was also applied to training data with large differences in sample sizes. In all cases, the same model was used without any adjustment in the layers, nodes, and type of nodes. It's likely that the Residual Units are implicitly performing some kind of regularization that leads to such stability.

5.3 Early Crossing vs. Late Crossing

It is worthwhile to compare Deep Crossing with DSSM. Fig. 3 is the architecture of a modified DSSM using log loss as the objective function. The modified DSSM is more closely related to the applications of click prediction. It keeps the basic structure of DSSM on the left side of the green dashed line, but uses log loss to compare the predictions with real-world labels. DSSM allows two text inputs, each represented by its tri-letter gram vector. DSSM has the distinct property of delaying the feature interaction (or crossing) to the late stage of the forward computation. Before reaching the Cosine Distance node, the input features are fully embedded through multiple layers of transformations on two separate routes. In contrast, Deep Crossing adopts at most one layer of single-feature embedding, and

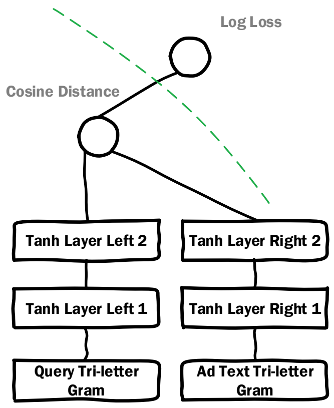


Figure 3: A DSSM Model with Log Loss

starts feature interaction at a much earlier stage of the forward computation.

DSSM and its convolutional variation (CDSSM [19]) have been proven by many real-world applications to be strong learning machines especially tuned for a pair of text inputs. Still, when comparing Deep Crossing and DSSM in Sec. 7.1 on two text only tasks, the former consistently outperforms the latter. Besides the superior optimization ability of the Residual Units, introducing the feature interaction early on in the forward computation seems to play an important role.

6. IMPLEMENTATION

Deep Crossing is implemented on a multi-GPU platform powered by the Computational Network Toolkit (CNTK)[1][3], which shares the same theoretical foundation of *computational network* with TensorFlow[4].

6.1 Modeling with CNTK

Building the Deep Crossing model is quite straight forward with CNTK. The script below defines a few Macros that will be used later.

```
ONELAYER(Dim_XO, Dim_XI, XI){
    W = Parameter(Dim_XO, Dim_XI)
    b = Parameter(Dim_XO)
    XO = Plus(Times(W, XI), b)
}
ONELAYERSIG(Dim_XO, Dim_XI, XI){
    t = ONELAYER(Dim_XO, Dim_XI, XI)
    XO = Sigmoid(t)
}
ONELAYERRELU(Dim_XO, Dim_XI, XI){
    t = ONELAYER(Dim_XO, Dim_XI, XI)
    XO = ReLU(t)
}
RESIDUALUNIT(Dim_H, Dim_XI, XI){
    l1 = ONELAYERRELU(Dim_H, Dim_XI, XI)
    l2 = ONELAYER(Dim_XI, Dim_H, l1)
    XO = ReLU(Plus(XI, l2))
}
```

The Macro `ONELAYER` creates a weight matrix `W` and a bias vector `b` for one layer of a network. Macros `ONELAYERSIG` and `ONELAYERRELU` apply element-wise Sigmoid and ReLU functions to the output of `ONELAYER` and build a full Sigmoid layer and a ReLU layer, respectively. Comparing side-by-side `ONELAYERRELU` and Equ. 2, `XI`, `XO`, `Dim_XI`, and `Dim_XO` are X_j^I, X_j^O, n_j, and m_j, respectively. The `RESIDUALUNIT` uses a hidden layer of `Dim_H`, but the output has the same dimension as `Dim_XI`. The script in Fig. 4 has the details of the actual Deep Crossing model when applied to the features in Table 1. As can be seen from the script, the full model has five steps. Each step has been described in Sec. 5. Since the MatchType feature `M` has only 4 dimensions, it goes directly to the stacking stage without embedding. This is why the embedding step and the stacking step are regarded as a single, combined step. `Dim_E` and `Dim_CROSS*` are the numbers of nodes in the embedding layer and the hidden layer inside the Residual Unit, respectively. `Dim_L=1` for log loss. Some details such as feature I/O and normalization are not included here for simplicity[5].

[3]http://www.cntk.ai
[4]https://www.tensorflow.org
[5]The full Deep Crossing model will be available at

```
## The Deep Crossing Model
## Step 1: Read in features, omitted
## Step 2A: Embedding
Q = ONELAYERRELU(Dim_E, Dim_Query, Query)
K = ONELAYERRELU(Dim_E, Dim_Keyword, Keyword)
T = ONELAYERRELU(Dim_E, Dim_Title, Title)
C = ONELAYERRELU(Dim_E, Dim_CampaignID, CampaignID)
## Step 2B: Stacking
# M = MatchType, CC = CampaignIDCount
Stack = RowStack(Q, K, T, C, M, CC)
## Step 3: Deep Residual Layers
r1 = RESIDUALUNIT(Dim_CROSS1, Dim_Stack, Stack)
r2 = RESIDUALUNIT(Dim_CROSS2, Dim_Stack, r1)
r3 = RESIDUALUNIT(Dim_CROSS3, Dim_Stack, r2)
r4 = RESIDUALUNIT(Dim_CROSS4, Dim_Stack, r3)
r5 = RESIDUALUNIT(Dim_CROSS5, Dim_Stack, r4)
## Step 4: Final Sigmoid Layer
Predict = ONELAYERSIG(Dim_L, Dim_Stack, r5)
## Step 5: Log Loss Objective
CE = LogLoss(Label, Predict)
CriteriaNodes = (CE)
```

Figure 4: A Deep Crossing Model with five layers of Residual Units, described in the CNTK modeling language

6.2 Multi-GPU Platform

The experiments were carried out on a GPU cluster that is optimized for CNTK for rapid, no-hassle, deep learning model training and evaluation. The cluster, which has high-throughput distributed storage, virtual file systems, and fault tolerance, is managed by specially designed automated cluster management and job/container scheduling software. During experimentation, each GPU machine in the cluster contained four K40 GPU cards. Infiniband is used to connect nearby GPU machines for high-speed data transmission between GPUs across machines.

To speed up the experiments, we have exploited the block-wise model-update filtering (BMUF) distributed training algorithm implemented in CNTK, originally proposed by Chen and Huo [2]. The BMUF algorithm improves over traditional model averaging (MA) and alternating direction method of multipliers (ADMM) while keeping the advantage of the low communication cost of these methods. While MA and ADMM can be easily scaled out, they often under perform single-GPU SGD with regard to the accuracy achievable with the trained model, and require different learning rate schedules than that of single-GPU SGD. The BMUF algorithm, however, does not have these drawbacks. Chen and Huo [2] reported a 28X speedup with 32 GPUs while also achieving better accuracy than the single-GPU SGD on large-scale speech recognition experiments. In our experiments reported in this paper, we reduced training time from *24 days* with a single-GPU, to *20 hours* by exploiting 32 GPUs across 8 machines.

7. EXPERIMENTATION

Deep Crossing was trained and evaluated on impression and click logs of a major search engine. Table 2 lists the data sets used for the experiments in this section. Each

Data Set	Group	Task	Type	Rows	Dims
text_cp1_tn_s	G_1	CP1	train	194	98.5
text_cp1_vd	G_1	CP1	valid	49	98.5
text_cp1_tt	G_1	CP1	test	45	98.5
text_cp1_tn_b	G_1	CP1	train	2,930	98.5
text_cp2_tn	G_2	CP2	train	518	98.5
text_cp2_vd	G_2	CP2	valid	100	98.5
text_cp2_tt	G_2	CP2	test	93	98.5
all_cp1_tn_s	G_3	CP1	train	111	202.7
all_cp1_vd	G_3	CP1	valid	139	202.7
all_cp1_tt	G_3	CP1	test	156	202.7
all_cp1_tn_b	G_3	CP1	train	2,237	202.7

Table 2: Data sets used in this paper, where Rows (in millions) is the number of samples in the data set, while Dims (in thousands) is the total number of feature dimensions

experiment will use a combination of training, validation, and test data sets, referred by the names in the "Data Set" column. Only data sets in the same "Group" are compatible, meaning that there is no overlap in time for different "Types". For example, a model trained with `all_cp1_tn_b` (the last row in the table) can be validated with `all_cp1_vd` and tested with `all_cp1_tt` since they all belong to the `G_3` group.

The "Task" of a data set is either `CP1` or `CP2`. The two tasks represent two different models in the click prediction pipeline. In this case, `CP1` is the more critical model of the two, however the models are complementary in the system.

The Deep Crossing model described in Fig. 4 is used for all the experiments. Model parameters are also fixed to 256, 512, 512, 256, 128, and 64 for `Dim_E`, and `Dim_CROSS[1-5]`, respectively. Readers are encouraged to experiment with these parameters including the number of layers for their specific applications to achieve optimal results.

Note that all experimental results comparing with baselines are statistically significant due to the sheer volume of samples used in the test data. We will not call out individually for simplicity.

7.1 Performance on a Pair of Text Inputs

As briefly mentioned in Sec. 5.3, we are interested in comparing DSSM and Deep Crossing in the setting that DSSM is specialized. To create an apples-to-apples comparison, we trained a DSSM and a Deep Crossing model on data for `CP1` and `CP2`, but limited the Deep Crossing model to the same data as DSSM (i.e., both use a pair of inputs that include query text and keyword or title text, each represented by a tri-letter gram vector).

In the first experiment, click prediction models were trained on task `CP1` on the two data sets listed in Table 3. On both data sets, Deep Crossing outperforms DSSM in terms of rel-

Training Data	DSSM	Deep Crossing
text_cp1_tn_s	100	100.46
text_cp1_tn_b	100	101.02

Table 3: Click prediction results for task CP1 with a pair of text inputs where performance is measured by relative AUC using DSSM as the baseline

https://github.com/Microsoft/CNTK/wiki/Examples. Please check the site for the latest.

Test Data	DSSM	Deep Crossing	Production
text_cp2_tt	98.68	98.99	100

Table 4: Click prediction results for task CP2 with a pair of text inputs where performance is measured by relative AUC using the production model as the baseline

ative AUC as explained in the table. Note that the DSSM model used here is the log loss version detailed in Sec. 5.3.

In the second experiment, both models were trained on task `CP2` on the `text_cp2_tn` dataset, and tested on the `text_cp2_tt` dataset. Table 4 shows the performance results for DSSM, Deep Crossing, and the model currently running inside our production system. The production model is trained on a different data set, but is tested with the same data (`text_cp2_tt`) using the prediction output logged during run-time. It can be seen that Deep Crossing performs better than DSSM but worse than the production model. This is expected since the production model uses many more features – including combinatorial features – and has been refined over many years. Still, by just using individual query and title features, Deep Crossing is only about one percentage point away from the production model.

While we have demonstrated Deep Crossing's ability to learn from simple pairs of text inputs, this is not its main goal. The real power of Deep Crossing is to work with many individual features, as we will see in the subsequent experiments.

7.2 Beyond Text Input

We now consider the performance of Deep Crossing on task `CP1` (training set `all_cp1_tn_s`) with about two dozen features[6] including those listed in Table 1. The experiments in this sub-section don't have external baselines, instead we will only compare Deep Crossing's performance with different combinations of features. Instead of assessing the performance relative to other methods, the goal here is to see Deep Crossing in action, and demonstrate its performance can change significantly as features are added and removed. We will compare the performance of Deep Crossing with production models in the next sub-section, using the same rich set of features.

In the first experiment, we ran Deep Crossing several times with different sets of features turned on and off. Counting features (see Sec. 4 for definition) are always turned off, and will be turned back on in the next experiment. The `All_features` model in Fig. 5 has all the remaining features turned on. As expected, it has the lowest log loss among all the models in this experiment. The `Only_Q_K` model has the highest log loss. This model uses only query text and keyword text, which is similar to the models in the previous sub-section.

Note that the log loss in Fig. 5 is *relative log loss* defined as the actual log loss divided by the lowest log loss of the `All_features` model over all epochs. The gap between `Only_Q_K` and `All_features` in terms of relative log loss is about 0.12. This is roughly a $7-8\%$ improvement in terms of AUC, which is huge given that an improvement of $0.1-0.3\%$ in AUC is usually regarded as significant for click prediction

[6]Unfortunately we were not able to disclose the full feature set but we did our best to make sure the lack of such details has minimal impact on the discussions of the results.

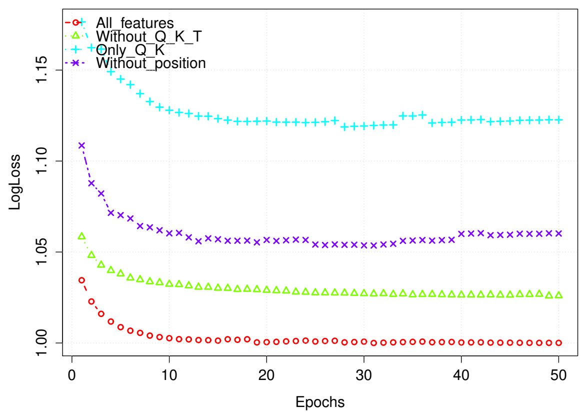


Figure 5: Deep Crossing's performance on task CP1 with different individual feature sets where performance is measured by relative log loss on the validation set over the training epochs (relative log loss is defined in Sec.7.2)

models. `Without_Q_K_T` is a model with query text, keyword text, and title text removed, which means that taking away the majority of text features will increase relative log loss by 0.025. This is a half of the increase with the model `Without_position`, which removes the position feature.

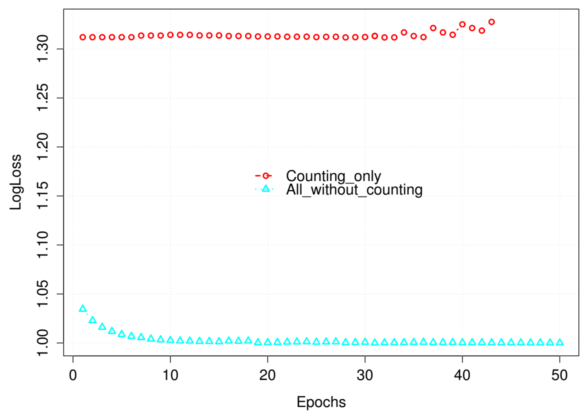
In the second experiment, we study how the counting features interact with the rest of the individual features. As discussed in Sec. 4, counting features play an important role in reducing the dimension of high cardinality features. Our full feature set has five types of counting features. In this experiment we just turned on one of them (referred as *the selected counting feature* hereafter) to demonstrate the effect.

As can be seen from Fig. 6a, the model `Counting_only` with only the selected counting feature is very weak compared with the `All_without_counting` model, which has all of the features except the counting features. Fig. 6b shows the result after adding the selected counting feature, where the new model `All_with_counting` reduced the relative log loss by 0.02. The base point for the relative log loss for both Fig. 6a and Fig. 6b is the lowest log loss of the `All_without_counting` model over all epochs.

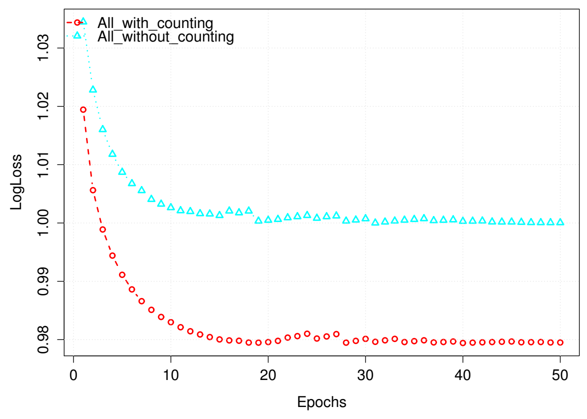
7.3 Comparison with Production Models

Up to this point, the question remains whether Deep Crossing can actually beat the production model, which is the ultimate baseline. To answer this question, we trained a Deep Crossing model with 2.2 billion samples, using a subset of the raw features from the production model. As reported in Table 5, the new model has comfortably outperformed the production model in offline AUC on task CP1 (results for task CP2 are not available as of the time of publication). It was trained on the task `CP1` (`all_cp1_tn_b`) data set , and tested on `all_cp1_tt`. The production model was trained on a different (and bigger) data set but was tested with the same data. As in the comparison with the task `CP2` model in Sec. 7.1, the production AUC was based on the prediction output logged during the run-time.

The above result is quite significant since the Deep Cross-



(a) A model (in red) using the selected counting feature only is weak



(b) Adding the selected counting feature reduced log loss significantly

Figure 6: The effect of adding the selected counting feature on task CP1 in Deep Crossing where performance is measured by relative log loss on the validation set over training epochs (relative log loss is defined in Sec.7.2)

Test Data	Deep Crossing	Production
all_cp1_tt	101.02	100

Table 5: Click prediction model compared with the production model on task CP1 where performance is measured by relative AUC using production model as the baseline

ing model used only a fraction number of features, and took much less efforts to build and maintain.

8. CONCLUSIONS AND FUTURE WORK

Deep Crossing enables web-scale modeling without extensive feature engineering. It demonstrated that with the recent advance in deep learning algorithms, modeling language, and GPU-based infrastructure, a nearly dummy solution exists for complex modeling tasks at large scale. While such claims require further testing, it does resonate with a key benefit of deep learning envisioned by the pioneers, which is to free people from the tedious work of feature engineering.

Deep Crossing was initially developed in the context of paid search ads, which is a web-application with massive data. When applied to other domains, we expect that most properties will still hold. This is because of the extensive coverage of feature types and the general model representations.

Besides saving effort in building large models, Deep Crossing also helps to decouple the application domains with modeling technologies. To hand craft combinatorial features requires extensive domain knowledge. But if modeling complexity is shifted from feature engineering to modeling technologies, the problem is opened up to the entire world of modeling experts, even those not working in the same domain. As evidence, the BMUF [2] distributed training algorithm developed by a few researchers in speech recognition helped Deep Crossing to fill the last bit of performance gap with a production model for sponsored search. While we don't have an explicit economic study, the compound effect of less efforts in feature engineering and getting more help from others is expected to be significant for a broad range of applications.

Deep Crossing is by design a multi-way fusion engine for heterogeneous features. As we apply it to more application domains, it will be interesting to see how this aspect is fully manifested.

9. ACKNOWLEDGMENTS

The authors would like to thank Chris Basoglu, Yongqiang Wang, Jian Sun, Xiaodong He, Jianfeng Gao, and Qiang Huo for their support and discussions that benefited the development of Deep Crossing.

10. REFERENCES

[1] A. Agarwal, E. Akchurin, C. Basoglu, G. Chen, S. Cyphers, J. Droppo, A. Eversole, B. Guenter, M. Hillebrand, T. R. Hoens, X. Huang, Z. Huang, V. Ivanov, A. Kamenev, P. Kranen, O. Kuchaiev, W. Manousek, A. May, B. Mitra, O. Nano, G. Navarro, A. Orlov, M. Padmilac, H. Parthasarathi, B. Peng, A. Reznichenko, F. Seide, M. L. Seltzer, M. Slaney, A. Stolcke, H. Wang, Y. Wang, K. Yao, D. Yu, Y. Zhang, and G. Zweig. An introduction to computational networks and the computational network toolkit. Technical report, Microsoft Technical Report MSR-TR-2014-112, 2014.

[2] K. Chen and Q. Huo. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Internal Conference on Acoustics, Speech and Signal Processing*, 2016.

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[4] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks

for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[5] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Technical report, National Bureau of Economic Research, 2005.

[6] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics*, 36:193–202, 1980.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[9] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[10] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. IEEE, 1999.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[15] S. Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[16] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[17] F. Seide, G. Li, X. Chen, and D. Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 24–29. IEEE, 2011.

[18] F. Seide, G. Li, and D. Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pages 437–440, 2011.

[19] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 373–374. International World Wide Web Conferences Steering Committee, 2014.

[20] D. Yu and L. Deng. Deep learning and its applications to signal and information processing [exploratory dsp]. *Signal Processing Magazine, IEEE*, 28(1):145–154, 2011.

[21] D. Yu, G. Hinton, N. Morgan, J.-T. Chien, and S. Sagayama. Introduction to the special section on deep learning for speech and language processing. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):4–6, 2012.