Locally Connected Deep Learning Framework for Industrial-scale Recommender Systems

Cen Chen\textsuperscript{1,2}\textsuperscript{*}, Peilin Zhao\textsuperscript{2}, Longfei Li\textsuperscript{2}, Jun Zhou\textsuperscript{3}, Xiaolong Li\textsuperscript{2}, and Minghui Qiu\textsuperscript{3}
\textsuperscript{1}Singapore Management University, Singapore
\textsuperscript{2}Ant Financial Group, Hangzhou, China
\textsuperscript{3}Alibaba Cloud, Hangzhou, China
\textsuperscript{*}cenchen@smu.edu.sg,\textsuperscript{2,3}{peilin.zpl,longyao.1lf,jun.zhoujun,xl.li,minghui.qhm}@alibaba-inc.com

ABSTRACT
In this work, we propose a locally connected deep learning framework for recommender systems, which reduces the complexity of deep neural network (DNN) by two to three orders of magnitude. We further extend the framework using the idea of recently proposed Wide&Deep model. Experiments on industrial-scale datasets show that our methods could achieve good results with much shorter runtime.

Keywords
DNN; Locally-Connected DNN; Wide&Deep

1. INTRODUCTION
In many real-world recommender systems, most of the features are categorical, which are usually discretized into one-hot representation. The input feature size after discretization and feature crossing can easily reach millions or even billions. Table 1 summarizes the real-world datasets we used, where more than 99.9% are sparse features. Despite the compelling benefits of deep learning models, it is challenging to scale up to industrial size, due to the largely increasing parameter space. If we could find an effective way to handle the feature sparsity issue and compress the model, implementing deep learning models in the industry could be a promising option. In this work, we propose a general locally connected deep learning framework to address large-scale industrial-level recommendation task, that transforms the one-hot sparse features into dense input and significantly reduces the model size. We tested the framework on several industrial-scale datasets and deployed it on Alipay recommendation systems. Experiments show our methods could achieve good results within a shorter amount of time.

2. MODEL ARCHITECTURE
Logistic Regression (LR) has been widely used in industrial-scale recommender systems, due to its simplicity and high scalability. In this section, we first introduce the LR and its equivalent, followed by presenting a locally-connected DNN that efficiently resolves the sparsity and scalability issues.

2.1 Logistic Regression and its Equivalent

LR trains a regression model, i.e., $\hat{y} = \sigma(W^T x + b)$, to fit the input feature vector $x$ and label $y$, where $\sigma$ is the logistic function. LR model can be transformed into an equivalent 3-layer model shown on the right side of Figure 1. This transformation process decomposes the weight matrix $W$:

$$ W = M_1 \times M_2, $$

where $M_1 = \text{Diag}(W_{\text{Feat}_1}, W_{\text{Feat}_2}, W_{\text{Feat}_3})$ $= \text{Diag}([w1, w2], [w3, w4], [w5, w6, w7, w8])$.

$M_1 \in \mathbb{R}^{d \times f}, M_2 \in \mathbb{R}^{f \times 1} (d = 8, f = 3)$.

$M_1$ and $M_2$ are the parameters for the equivalent 3-layer model, where $M_1$ is a sparse ‘diagonal’ matrix with only $d$ values and $M_2$ is a dense vector of value ones. We refer this equivalent model as a locally-connected network. In the next section, we expand this idea to the case of DNN.

2.2 Locally Connected DNN

DNN [3], that hierarchically learns the high-level abstractions from low-level features through multiple layers of non-linear transformation, can significantly outperform LR over various tasks. For the ease of explanation, we first present a four-layer DNN in Figure 4(a).

$$ \hat{y} = g_2(W_3^T l_2 + b_3), \quad l_2 = g_1(W_2^T l_1 + b_2), \quad l_1 = g_1(W_1^T x + b_1), $$

$$ l_0 = M_1^T x, $$

(a) 4-layer DNN  \quad (a) 5-layer locally connected DNN

Figure 4: 4-layer DNN VS. 5-layer locally connected DNN.

\textsuperscript{*}Work done during the internship at Ant Financial Group.

\textsuperscript{1}Example here ‘diagonal’ w.r.t. three feature groups.
where $g$ is the non-linear activation function; $W_1 \in \mathbb{R}^{d \times K_1}$, $W_2 \in \mathbb{R}^{K_1 \times K_2}$, $W_3 \in \mathbb{R}^{K_2 \times O}$ are the model parameters. Here, $K_1$ and $K_2$ denote the parameter sizes for the hidden layers. Let $f$ be the original feature space and $d$ be the discretized feature space. Typically for industry dataset, we have $f \ll d$, by several orders of magnitude, as categorical features, such as user_id, can easily be expanded to a few million dimensions. $f$, $K_1$, and $K_2$ are usually in a few hundreds or thousands at most. Thus the model size of a DNN largely depends on $W_1$. Similar to Eqn. 1, we decompose matrix $W_1$ as $W_1 = M_1 \times M_2$, where $M_1 \in \mathbb{R}^{d \times f}$, $M_2 \in \mathbb{R}^{f \times K_1}$.

We further compress the DNN into a 5-layer locally connected DNN model, (LC-DNN), outlined in Figure 4(b) and Figure 5. As shown in Figure 1, $M_1$ can be pre-trained by LR. In the actual implementation, we first train an LR on the discretized features. We then feed LC-DNN the learned LR weights, which are jointly trained with other parameters.

**Extension to Wide & Deep:** Inspired by the work of [1], we further extend our approach to Wide & Deep framework, referred as LC-W&D, where a linear model (wide component) is jointly trained with LC-DNN (deep component).

**Model complexity:** We compare the parameter sizes of different models in Table 2. As $f$, $K_1$, $K_2 \ll O(d)$, our method can significantly reduce the parameter space of DNN.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameter Size</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>$O(d)$</td>
<td></td>
</tr>
<tr>
<td>4L DNN</td>
<td>$O(dK_1) + O(K_1K_2)$</td>
<td>$O(dK_1)$</td>
</tr>
<tr>
<td>5L LC-DNN</td>
<td>$O(d) + O(fK_1) + O(K_1K_2)$</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>5L LC-W&amp;D</td>
<td>$O(2d) + O(fK_1) + O(K_1K_2)$</td>
<td>$O(2d)$</td>
</tr>
</tbody>
</table>

Table 2: A comparison of parameter sizes.

Note that our method effectively reduces the input feature space which makes the training of a deep neural network easier. The same method can also be applied to deep residual network [2] to build a deeper network.

3. EXPERIMENTS

We test the performance of our proposed approaches against LR and DNN on two real-world datasets. Datasets used are summarized in Table 1. All the models are implemented on our own implementation of the parameter server [4] and trained over the cluster.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Training</th>
<th>Testing</th>
<th>#RawFeat</th>
<th>#InputFeat</th>
<th>#CategoricalFeat</th>
<th>FeatureSparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppData</td>
<td>App recommendation</td>
<td>9,000,000</td>
<td>2,035,501</td>
<td>158</td>
<td>8,672,633</td>
<td>8,672,607</td>
<td>99.997%</td>
</tr>
<tr>
<td>FeedsData</td>
<td>Feeds ranking</td>
<td>86,980,291</td>
<td>23,249,245</td>
<td>2,311</td>
<td>8,522,742</td>
<td>8,521,329</td>
<td>99.9834%</td>
</tr>
</tbody>
</table>

Table 1: Dataset description and statistics.

We further compress the DNN into a 5-layer locally connected DNN model, (LC-DNN), outlined in Figure 4(b) and Figure 5. As shown in Figure 1, $M_1$ can be pre-trained by LR. In the actual implementation, we first train an LR on the discretized features. We then feed LC-DNN the learned LR weights, which are jointly trained with other parameters.

Figure 5: A 5-layer locally connected DNN (LC-DNN).

**Parameter Size:** We first theoretically compare the parameter sizes of LR, four-layer DNN with the network structure of $[N_{input}, 2048, 1024, N_{output}]$, and the corresponding five-layer LC-DNN on our datasets. Clearly, LC-DNN reduces the parameter size of DNN by three orders of magnitude.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LR</th>
<th>DNN</th>
<th>LC-DNN</th>
<th>LC-W&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppData</td>
<td>8.67E+6</td>
<td>1.78E+10</td>
<td>1.11E+7</td>
<td>1.98E+7</td>
</tr>
<tr>
<td>FeedsData</td>
<td>8.52E+6</td>
<td>1.75E+10</td>
<td>1.54E+7</td>
<td>2.39E+7</td>
</tr>
</tbody>
</table>

Table 3: Parameter size comparison.

**Solution quality:** We use area under receiver operator curve (AUC) to measure the solution quality. From Table 4, we observe that LC-DNN and LC-W&D achieve higher AUC than both LR and DNN.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>AUC</th>
<th>LR</th>
<th>DNN</th>
<th>LC-DNN</th>
<th>LC-W&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppData</td>
<td>0.9131</td>
<td>0.8633</td>
<td>0.9205</td>
<td>0.9207</td>
<td></td>
</tr>
<tr>
<td>FeedsData</td>
<td>0.7835</td>
<td>0.7847</td>
<td>0.7905</td>
<td>0.7918</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Offline AUC performance on testing sets.

**Runtime:** Figure 6 records the average training time per epoch, in seconds. Figure 6 shows that our proposed methods improve the runtime of DNN by one order of magnitude. Due to the auto load-balance and communication cost of the distributed implementations over the cluster, runtime reductions are not as prominent as the theoretical estimations.

![Figure 6: Runtime comparison.](image)

4. CONCLUSIONS

In this work, we propose a locally connected deep learning framework for recommender systems, which reduces the model complexity of DNN by several orders of magnitude. We further extend the framework using the idea of the Wide & Deep model. Experiments show that our proposed methods could achieve good results with much shorter runtime.

5. REFERENCES


