Visualizing and Understanding Deep Neural Networks in CTR Prediction

Lin Guo
Alibaba Group

Hui Ye
Alibaba Group

Wenbo Su
Alibaba Group

Henhuan Liu
Alibaba Group

Kai Sun
Alibaba Group

Hang Xiang
Alibaba Group

ABSTRACT

Although deep learning techniques have been successfully applied to many tasks, interpreting deep neural network models is still a big challenge to us. Recently, many works have been done on visualizing and analyzing the mechanism of deep neural networks in the areas of image processing and natural language processing. In this paper, we present our approaches to visualize and understand deep neural networks for a very important commercial task—CTR (Click-through rate) prediction. We conduct experiments on the productive data from our online advertising system with daily varying distribution. To understand the mechanism and the performance of the model, we inspect the model's inner status at neuron level. Also, a probe approach is implemented to measure the layer-wise performance of the model. Moreover, to measure the influence from the input features, we calculate saliency scores based on the back-propagated gradients. Practical applications are also discussed, for example, in understanding, monitoring, diagnosing and refining models and algorithms.

ACM Reference format:
Lin Guo, Hui Ye, Wenbo Su, Henhuan Liu, Kai Sun, and Hang Xiang. 2018. Visualizing and Understanding Deep Neural Networks in CTR Prediction. In *Proceedings of ACM SIGIR Workshop on eCommerce, Ann Arbor, Michigan, USA, July 2018 (SIGIR 2018 eCom)*, 7 pages.
https://doi.org/

1 INTRODUCTION

Click-through rate (CTR) prediction plays a crucial role in computational advertising. In the common cost-per-click advertising system, advertisements are ranked by the product of the bid price and the predicted CTR when bidding for impression opportunities. Therefore, the revenue of the multi-billion business heavily relies on the performance of the CTR prediction model.

Deep learning techniques have been successfully applied to CTR prediction tasks [6, 7, 23]. Deep neural networks (DNNs), composed of stacked layers of neurons, have the capability to extract the nonlinear patterns from features and thus reduce the burden of nontrivial feature engineering. However, the working mechanisms of deep learning models are still not well understood. The lack of interpretability becomes an obstacle for deep learning, and raises concerns on the reliability of deep learning applications, especially for critical industrial implementations.

Many recent progresses have been made in visualizing and interpolating deep learning models for image processing [15, 18, 20, 21, 26, 29] and natural language processing [3, 4, 14, 16, 27]. In this paper, we present a series of approaches to visualize and analyze a simple DNN model for CTR prediction on the productive data from our search advertising platform. The model's performance decay is investigated over datasets with daily varying distribution, and the distributions of the output scores are also compared for different training stages. We inspect the model's inner status down to neuron level. We study the statistical properties of the neurons' statuses for the hidden layers, and investigate the high-level representations learned by the model through t-SNE projection [17, 21]. A probe method [2] is applied to dissect model's performance layer by layer for different datasets. Moreover, to measure the influence of the input features, we calculate saliency scores for the feature groups based on back-propagated gradients.

Beyond the classic model evaluation metrics [11, 12], we open up the "black box" and inspect the DNN model from the output to the input end. Understanding the model's mechanism can help us not only design and diagnose models, but also monitor the algorithmic advertising system for daily production.

2 EXPERIMENTAL SETTING

2.1 Datasets

We perform experiments on the productive CTR prediction data from the search advertising platform of our company. Started from a typical Wednesday, our data are collected over eight consecutive days. The training set is sampled from day one. To investigate decay of the model's performance, we evaluate the model on a daily basis from day one to day eight. The eight test sets are, in turn, denoted by test1, test2, ..., test8. Each dataset contains about 150 million instances which are randomly sampled from the ad impression logs of the corresponding day. Note that there are no overlap between test1 and the training set. The setup of datasets simulates the real world environment for the CTR prediction task, i.e., the model is trained with historical data and deployed to serve the future online traffic, where the data distribution varies and differs with the training data by nature.

Our data contains 34 groups of sparse categorical features (around 100 million binary features in total), e.g., user id, user's city, user's gender, user's age level, query id, query words, shop id, ad's category, etc.. Note that there are no combinational features in this study.

In: J. Degenhardt, G. Di Fabbrizio, S. Kallumadi, M. Kumar, Y.-C. Lin, A. Trotman, H. Zhao (eds.): Proceedings of the SIGIR 2018 eCom workshop, 12 July, 2018, Ann Arbor, Michigan, USA, published at http://ceur-ws.org

2.2 Model setting

The DNN model contains four fully-connected hidden layers. From layer 1 (closest to input) to layer 4 (right before output layer), the layer's width is set to 256, 128, 64 and 32 neurons. The formulation for the output vector of kth hidden layer, denoted by $\mathbf{h}_k$, can be written as:

$$\mathbf{h}_k = ReLU(\mathbf{W}_k\mathbf{h}_{k-1} + \mathbf{b}_k), \quad (1)$$

Where $\mathbf{W}_k$ is the weight tensor of all the connections from the neurons of layer $(k-1)$, $\mathbf{b}_k$ represents the bias term and ReLU (rectifier linear unit) function is used as the activation function. The output layer uses a sigmoid function to map the output to a float number between 0 and 1 as the predicted probability of click:

$$Pctr = Sigmoid(\mathbf{W}_5\mathbf{h}_4 + \mathbf{b}_5). \quad (2)$$

For the training process, $Pctr$ is compared against the ground truth label and cross entropy is calculated as the loss function. For each input instance, the sparse feature ids are embedded into 8-dimensional float vectors [6, 7, 23]. For feature groups containing multiple feature ids per instance, e.g., query words, sum pooling operations are applied to enforce each feature group to produce an 8-dimensional embedding vector. The embedding outputs are concatenated into a 272-dimensional vector, denoted by $\mathbf{h}_0$, as the input to layer 1. The embedding vectors are trained jointed with the other parts of the model.

The experiments are run on distributed TensorFlow [1] released by Google. The model is trained by Adagrad optimizer [8] with learning rate = 0.005, initial accumulator value = 0.0001 and mini-batch size = 1000. Glorot and Bengio's method [10] is used for initialization. We visualize the model's inner status by dynamically dumping the processing data based on model graph.

3 RESULTS

3.1 AUC and Prediction Score

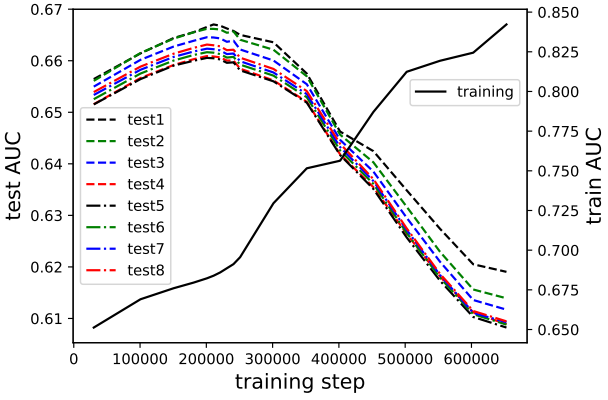

Figure 1: AUC score as a function of training step for training and test sets.

To measure the performance of model, we employ AUC (area under curve of the receiver operating characteristic plot) as the key metric. AUC is a widely used measure for evaluating the CTR performance [12].

In Fig. 1, we present the evolution of the model's AUC as a function of the training steps for training and test sets. With the training going on, the train AUC keeps growing, while all the test AUCs follow a same pattern — first rises and then decreases due to overfitting. The model generalizes best at step 210000. Comparing the eight test AUCs for the same time step, the model's performance decay can be disclosed as a function of dataset. The test AUC score decreases monotonically from day one to day five. As expected, this is because the distribution of the test data differs with the training set, and the difference grows day by day. After that, AUC upswings for the last three days and surpasses day four. This is in accordance with a characteristic of our business scene — although the data varies from day to day, the users' behaviors on our website have weekly periodic patterns. This non-monotonic change of AUC is evident for the regime from under-fitting to weak overfitting (before step ~ 400000). At larger training steps, overfitting becomes severe and the model performs same bad for the last five days.

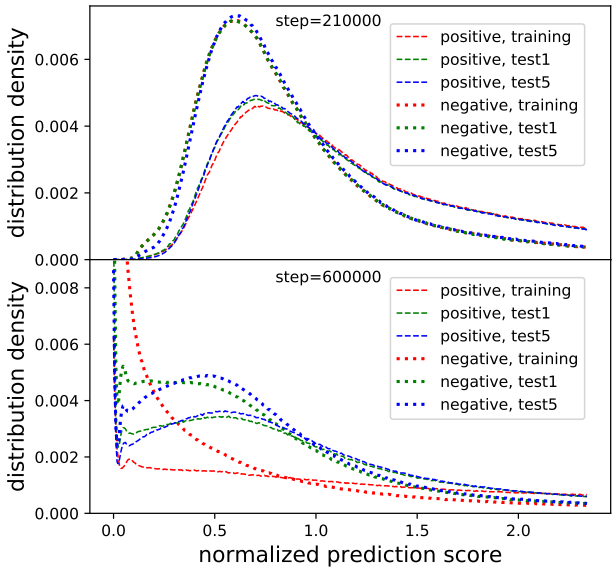

Figure 2: Distribution of predicted CTR for models at training step 210000 and step 600000. The X-axis denotes the predicted CTR normalized by the average click ratio of the training set.

Fig. 2 provides insights into the distribution of predicted CTR score for training, test1 and test5 sets. At training step 210000, the AUC decay from training set to test1 is mainly because the CTR of the positive (clicked) samples in test1 are more under-predicted by the model. The further decay from test1 to test5 is mainly due to that the negative (non-clicked) samples in test5 tend to be predicted with higher CTRs (the train and test1 curves overlap for the negative samples and can hardly be distinguished by eye). For training step

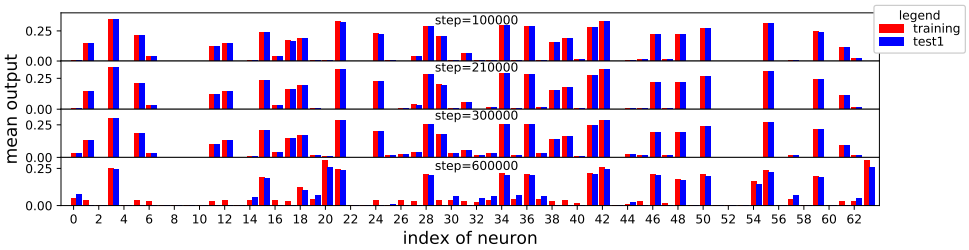

Figure 3: Mean outputs of the neurons in layer 3 for training and test1 sets, for training step 100000, 210000, 300000 and 600000. Each bar represents a neuron.

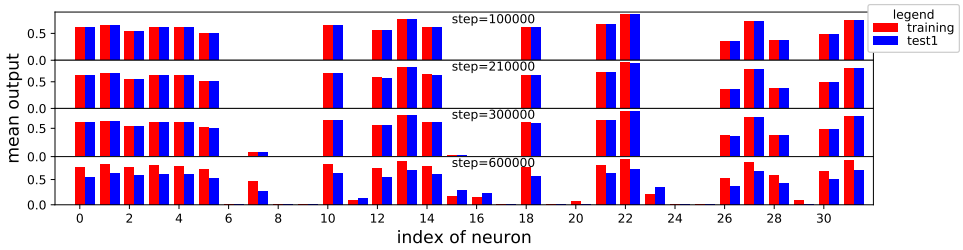

Figure 4: Mean outputs of the neurons in layer 4 for training and test1 sets, for training step 100000, 210000, 300000 and 600000. Each bar represents a neuron.

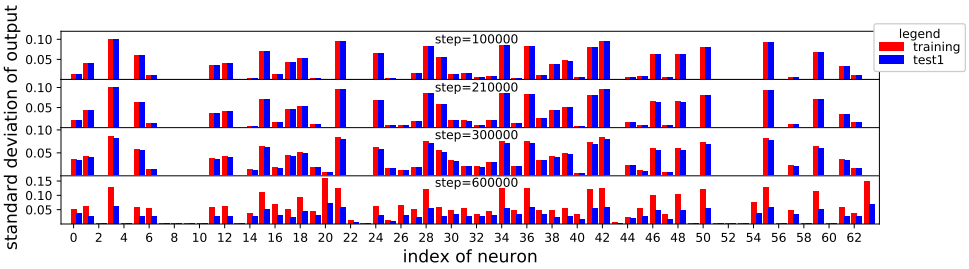

Figure 5: Standard deviations of the outputs of the neurons in layer 3 for training and test1 sets, for training step 100000, 210000, 300000 and 600000. Each bar represents a neuron.

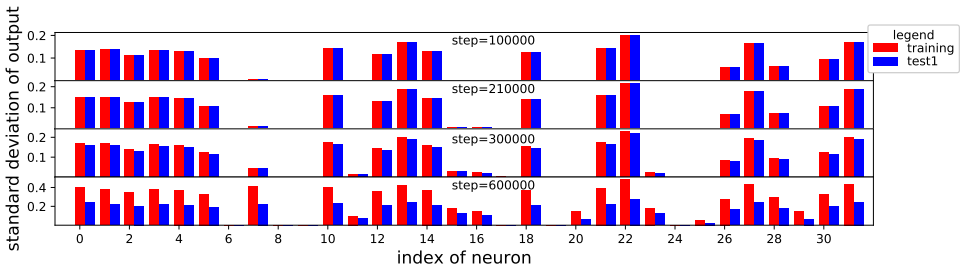

Figure 6: Standard deviations of the outputs of the neurons in layer 4 for training and test1 sets, for training step 100000, 210000, 300000 and 600000. Each bar represents a neuron.

600000, the model overfits the training data such that it aggressively predicts the CTR towards zero for both clicked and non-clicked samples. This is attributed to the high skewness of the data. The proportion of clicked samples is lower than 10%, so under-predicting the CTR for all samples may still reduce loss in training. This shape

of distribution changes significantly as the data become different, the scores move rightwards and the distribution becomes blurred.

3.2 Neuron Status

In this subsection, we investigate the statistics of the neurons' statuses for different training stages and datasets. These statistical properties depict the model's representation of the input data, and can help us to interpret the model's performance and working mechanism.

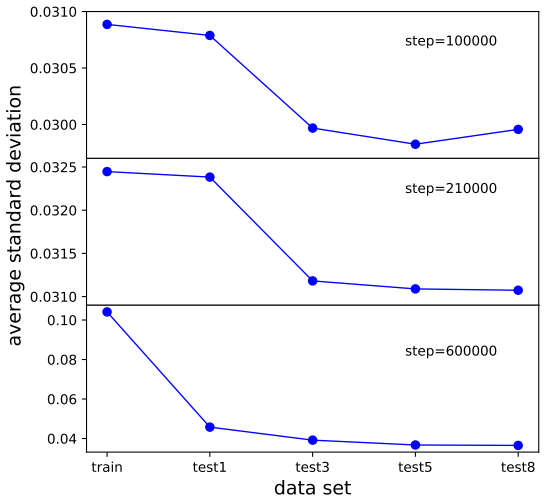

Figure 7: Average standard deviation of neurons' outputs for layer 3 as a function of dataset, for training step 100000, 210000 and 600000. The output's standard deviation is first calculated for each neuron, and then averaged over all the 64 neurons of layer 3.

The mean outputs of the neurons within layer 3 and 4 are illustrated in Figs. 3 and 4, respectively. Correspondingly, the standard deviation of the neurons' outputs are plotted in Figs. 5 and 6. For step 100000 and 210000, the results are quite close between the underfitting and well-fitting stages. About a quarter of the neurons are barely activated. Significant changes are observed for the overfitting regime (step > 300000). More neurons become activated. Also, the difference between the training and test sets grows with the degree of overfitting, especially in the standard deviation (Figs. 5 and 6). The higher standard deviation on the training set indicates that the neurons become over sensitive to the input of the training data. Fig. 7 presents the variation of the standard deviation averaged over all the 64 neurons of layer 3 as a function of dataset. For all the three different training stages, the trend of the average standard deviation correlates with the model's AUC score (Fig. 1).

To gain more knowledge about the collaborative patterns of neurons inside the model [21, 26], for each layer, we calculate the correlations among the neurons. Neurons' statuses before activation are used. We measure the average degree of neurons' correlations by averaging the absolute value of all the correlation coefficients for each layer. The average strength of correlations is plotted as a function of training step in Fig. 8. The degree of correlation climbs

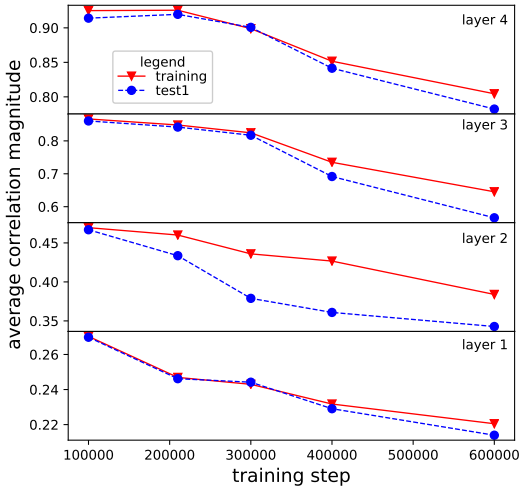

Figure 8: Average magnitude of the correlations among the neurons for each hidden layer. The evolution as a function of training step is plotted for training and test1 set.

up with the height of layer. This indicates that the DNN model is refining the input information through the successive layers [22, 24, 28]. Only very limited portion of the input information can be transfered to the output layer.

After step 210000, the neurons' correlation deceases monotonically with training step for all hidden layers. Recalling the enhanced neuron activation observed for this overfitting regime (Figs. 3 and 4), we can interpret that the model starts to explore more predictive patterns from the input information. However, the deceasing test AUC (Fig. 1) reveals that the boosted representation of the input from training data can not be well generalized to predict the test data.

In order to inspect the spacial structure of the high-level representations for the input data, we project the neurons' output vectors to 2-dimensional space using t-SNE method [17, 21]. The t-SNE projection is able to preserve neighborhoods and clusters of the data points in the original representation space. In Fig. 9, we illustrate the projection results for layer 2, 3 and 4 at training step 210000. The presented 10000 clicked and 10000 non-clicked instances are randomly selected from the training set.

For layer 3 (the center plot in Fig. 9), we can clearly see the regions with concentrated clicked points. We find that the training process enhances the concentration of clicked points for the training set, indicating that the model learns more discriminative representation for the training data. For the test datasets, we observe that the concentrated distribution disappears when overfitting happens. Unlike the case of image classification in Ref. [21], no class separation is observed even at severely overfitting stage. This is mainly due to the highly noisy and skewed data for the CTR prediction task.

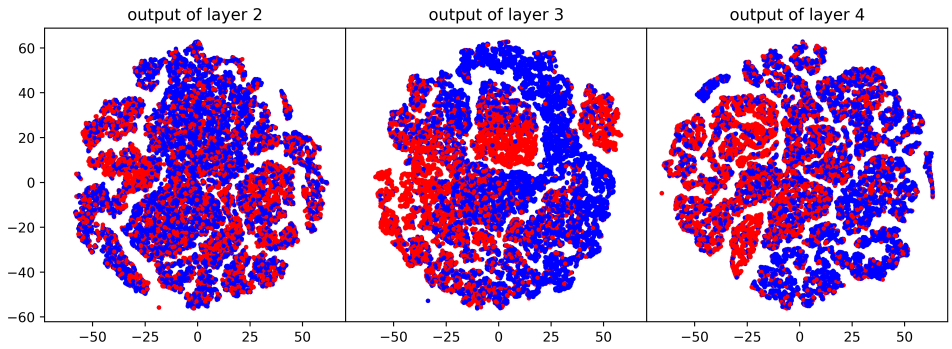

Figure 9: Visualization of the output vectors for layer 2, 3 and 4 using t-SNE method, at training step 210000. Clicked and non-clicked samples are represented by red and blue points, respectively

Comparing with the left plot in Fig. 9, the concentration of clicked points of layer 2 is obviously worse than layer 3. This agrees with the assumption that for a properly trained DNN model, the discriminative quality of the hidden layer's output increases with the height of the layer [2, 5, 21]. However, as revealed in the right plot of Fig. 9, the clicked points for layer 4 show no improvement in the degree of concentration and look even slightly more scattered. Recalling the very strong correlations among the neurons in layer 4 (Fig. 8), one may doubt whether the output of layer 4 is more predictive than layer 3. This issue will be further discussed in the following subsections.

3.3 Probe Evaluations

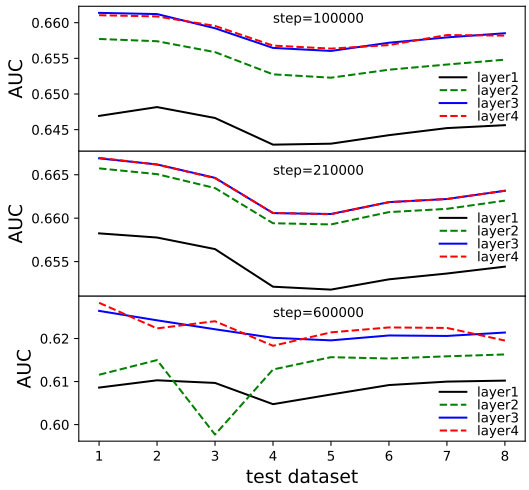

Figure 10: Test AUC scores of the probe LR models as a function of test dataset for three training steps: 100000, 210000 and 600000.

To investigate the effectiveness of the hidden layers, we implement Alain & Bengio's probe approach [2]. DNN model is expected to mining for predictive patterns from input features through layers of transformations, and then feed the extracted information into the simple linear classifier at the output end. For each layer, we use the layer's output vector as input features to train a LR (Logistic Regression) model to predict CTR. The LR model serves as a probe to evaluate the usefulness the hidden layer. A higher performance of the LR probe implies that the transformation of this layer makes information more predictive, and thus benefits the performance of the whole DNN model.

The LR models are trained on the data of the training set until convergence, with the DNN model fixed, and then the performances are evaluated on the tests sets. As shown in Fig. 10, for training step 210000, the performance increases from layer 1 to layer 3, indicating that these layers do transform input information to be more predictive. The probe's performance for layer 4 is the same as layer 3, indicating that layer 4 is not as useful as the previous three layers. This is consistent with the observations in the last subsection.

The change of AUC along each curve (in Fig. 10) illustrates how the hidden layer reacts to the varying data distribution. At training step 210000 where the DNN model generalizes best, the effectiveness of all the layers varies as a function of dataset in the same pattern with the DNN model. In contrast, for training step 100000, where the DNN model is underfitting, layer 1 behaves differently with the other layers. Moreover, for step 600000, the DNN model overfits the training data such that the learned information transformations begin to fail for test data. Therefore, the performance of probes is very low and fluctuates significantly.

3.4 Feature Group Saliency

For the input end of the DNN model, we study how the input features influence the model with the back-propagated gradient signals [16]. The embedding output of the sparse feature ids (concatenated as $\mathbf{h}_0$) can be treated as the input for the following deep neural network. With the model fixed, for each input instance, we calculate the gradient of $\mathbf{h}_0$ with respective to the model's output

$Pctr$:

$$\mathbf{g}_0 = \nabla_{\mathbf{h}_0} Pctr. \tag{3}$$

The magnitude of each element of the gradient vector $\mathbf{g}_0$ quantifies the sensitivity of the model's output to the change in the particular embedding element. It describe how much a small change in a particular embedding value could affect the final output $Pctr$. Given a dataset, we calculate the saliency score for each feature group by averaging the mean absolute value of the corresponding 8 gradient elements in $\mathbf{g}_0$ over the whole dataset. This saliency score provides us with an average measure of the model's sensitivity to each feature group for the given dataset.

We illustrate the saliency scores in Fig. 11. Overall, the model is becoming increasingly sensitive to all the feature groups during training. In the overfitting regime, the score of feature group 10 rises up dramatically and becomes much higher than the other feature groups. This feature group is composed of user ids, in which the number of ids is larger than any other feature group by at least two orders of magnitude [9]. For this training stage, the model is trained to memorize the vast amount of information from user ids that is not generalizable, and thus significantly deteriorates the performance on test datasets.

4 DISCUSSION

4.1 Role of Layer 4

The results about layer 4 raise a question about the necessity to include this layer in the model. To answer this question, we modify the neural network and investigate the impact on performance of the retrained models. We modify layer 4 by reducing or increasing its width by a factor of two, or even remove layer 4 from the model. It turns out that these modifications do not affect the models' performance (highest test AUCs) for the different test dataset. Although not harmful, there is no benefit to include layer 4 in the DNN model.

4.2 Regularization

Analysis in the previous section reveals that the model become over sensitive to the input when overfitting. Also, the high correlations among neurons for layer 3 and 4 (Fig. 8) imply that there might be severe co-adaptations [25]. One may hope to use regularizations to control overfitting and obtain better performance on test data. We have tried L1 and L2 regularization [11], and dropout [25], for a variety of hyper-parameters. However, no improvement is obtained. In future, more work needs to be conducted on improving model's generalization power.

4.3 Feature Treatment

Subsection 3.4 discloses the problem that the model is greatly sensitive to the feature group of user ids when overfitting. Other than regularization, it is also possible to improve the models' generalization power by optimizing the input features. User id is a highly granular feature group. Inputting it directly to the embedding-based deep neural network may not be the optimal choice. Following the idea of Wide&Deep [6], we remove user id from the embedding layer. The bias of each user id is represented by a float number b_{user} and added immediately into the output layer:

$$Pctr = Sigmoid(\mathbf{W}_5\mathbf{h}_4 + b_5 + b_{user}). \tag{4}$$

This bias is trained jointly with the other parts of the model. We find this approach can improve AUC on the test datasets by about 0.1%.

5 APPLICATIONS

With the visualization and analysis techniques presented above, we discuss some of the practical applications in this section.

- The distribution of the predicted CTR score is very important for real-time bidding auctions. Understanding the score distribution can help us to design better calibration methods [13, 19]. Also, score distribution can help to find outliers or bad-fitted samples, which can in turn be used to improve the model.
- Inspections of model's inner status and gradient signals open up the "black box" of the DNN model, helping us to understand the mechanism of the model and the influence of features. These approaches can be used to diagnose the model, like (but not limited to) underfitting/overfitting, gradient vanishing/explosion, ineffective model structure, etc.. A deep understanding of the model's mechanism can help us to design better model structure, training algorithm and features.
- For online advertisting, it is of great importance to monitor the model's online performance and the health of data pipeline. Feeding the model with problematic data can cause disaster. However, it is very difficult to describe and monitor the distribution of the extremely sparse and high-dimensional data. Moreover, monitoring the model's online performance may not be sufficient. The model predicts CTR for hundreds of candidate ads for each biding, while only very few ads can win the bidding and get feedback from impression. The classic performance metrics are mainly based on those feedbacks, and thus can only cover a limited portion of biased data.

  The DNN model, by nature, transforms the sparse input data into dense numerical representations. Therefore, the statistics of neurons' output and the gradient signals can be implemented as a new kind of metrics to monitor the distribution of the input data. Note that no feedback labels are needed to calculate these quantities. For example, as illustrated in Fig. 7, the average standard deviation for layer 3's output changes with the naturally varying distribution of input data. Problematic input data can cause more significant change in the statistics.

6 CONCLUSION

In this work, we visualize and analyze a simple DNN model for CTR prediction down to neuron level. Model training and evaluations are performed over a series of datasets. The model is inspected from the output to the input end. The statuses of neurons are studied using a variety of methods. Gradients of the feature embeddings are used to create a salience map to describe the influence of the feature groups. The analysis provides insightful knowledges of the model's mechanism, helping us to monitor, diagnose and refine the model.

Currently, we are applying these approaches to build a model-based evaluation and monitoring system for our online advertising

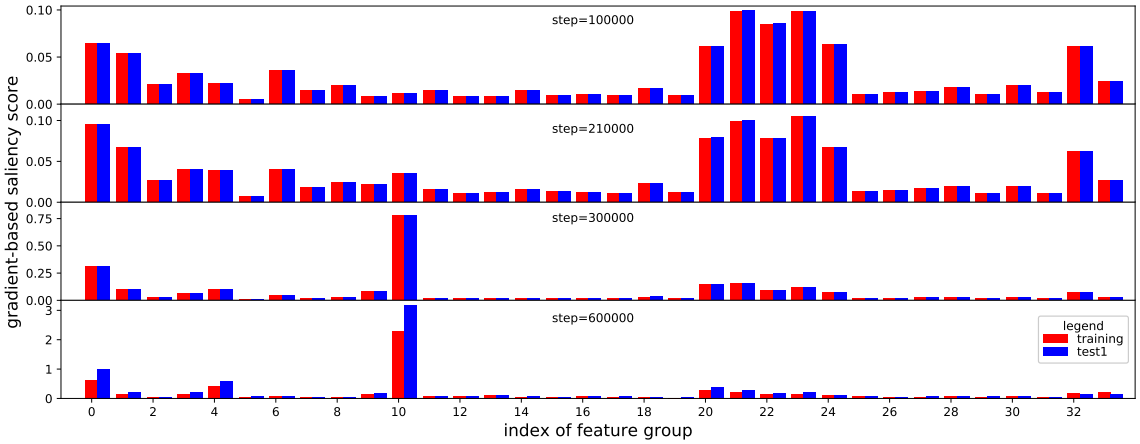

Figure 11: Gradient-based saliency score of the 34 feature groups for training and test1 sets. Each bar represents a feature group.

platform. Based on our industrial scenario, future work will focus on exploring more approaches to interpret deep learning, investigating more complex algorithms and applying these approaches to design better models and algorithms.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016). https://www.tensorflow.org/

[2] Guillaume Alain and Yoshua Bengio. 2016. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644* (2016).

[3] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. 2017. Explaining recurrent neural network predictions in sentiment analysis. *arXiv preprint arXiv:1706.07206* (2017).

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[5] Yoshua Bengio et al. 2009. Learning deep architectures for AI. *Foundations and trends® in Machine Learning* 2, 1 (2009), 1–127.

[6] Heng-Tze Cheng and Levent Koc. 2016. Wide & deep learning for recommender systems. In *Proceedings of the ACM 1st Workshop on Deep Learning for Recommender Systems*. 7–10.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of ACM Conference on Recommender Systems*. 191–198.

[8] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[9] Tiezheng Ge, Liqin Zhao, Guorui Zhou, Keyu Chen, Shuying Liu, Huiming Yi, Zelin Hu, Bochao Liu, Peng Sun, Haoyu Liu, et al. 2017. Image Matters: Jointly Train Advertising CTR Model with Image Representation of Ad and User Behavior. *arXiv preprint arXiv:1711.06505* (2017).

[10] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research* 9 (2010), 249–256.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[12] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale Bayesian Click-through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. Omnipress, USA, 13–20.

[13] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 1–9.

[14] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078* (2015).

[15] Pangwei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning*. 1885–1894.

[16] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and Understanding Neural Models in NLP. *arXiv preprint arXiv:1506.01066v2* (2016).

[17] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[18] Aravindh Mahendran and Andrea Vedaldi. 2016. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision* 120, 3 (2016), 233–255.

[19] Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1222–1230.

[20] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 1–18.

[21] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. 2017. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics* 23, 1 (2017), 101–110.

[22] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. 2018. On the Information Bottleneck Theory of Deep Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ry_WPG-A-

[23] Ying Shan and T Ryan Hoens. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of ACM Conference on Knowledge Discovery and Data Mining*.

[24] Ravid Shwartz-Ziv and Naftali Tishby. 2017. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810* (2017).

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[27] Zhiyuan Tang, Ying Shi, Dong Wang, Yang Feng, and Shiyue Zhang. 2017. Memory visualization for gated recurrent neural networks in speech recognition. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017).

[28] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*. 1–5. https://doi.org/10.1109/ITW.2015.7133169

[29] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.