Deep Learning over Multi-field Categorical Data – A Case Study on User Response Prediction

Weinan Zhang[1(✉)], Tianming Du[1,2], and Jun Wang[1]

[1] University College London, London, UK
{w.zhang,j.wang}@cs.ucl.ac.uk
[2] RayCloud Inc., Hangzhou, China
dutianming@quicloud.cn

Abstract. Predicting user responses, such as click-through rate and conversion rate, are critical in many web applications including web search, personalised recommendation, and online advertising. Different from continuous raw features that we usually found in the image and audio domains, the input features in web space are always of multi-field and are mostly discrete and categorical while their dependencies are little known. Major user response prediction models have to either limit themselves to linear models or require manually building up high-order combination features. The former loses the ability of exploring feature interactions, while the latter results in a heavy computation in the large feature space. To tackle the issue, we propose two novel models using deep neural networks (DNNs) to automatically learn effective patterns from categorical feature interactions and make predictions of users' ad clicks. To get our DNNs efficiently work, we propose to leverage three feature transformation methods, i.e., factorisation machines (FMs), restricted Boltzmann machines (RBMs) and denoising auto-encoders (DAEs). This paper presents the structure of our models and their efficient training algorithms. The large-scale experiments with real-world data demonstrate that our methods work better than major state-of-the-art models.

1 Introduction

User response (e.g., click-through or conversion) prediction plays a critical part in many web applications including web search, recommender systems, sponsored search, and display advertising. In online advertising, for instance, the ability of targeting individual users is the key advantage compared to traditional offline advertising. All these targeting techniques, essentially, rely on the system function of predicting whether a specific user will think the potential ad is "relevant", i.e., the probability that the user in a certain context will click a given ad [6]. Sponsored search, contextual advertising, and the recently emerged real-time bidding (RTB) display advertising all heavily rely on the ability of learned models to predict ad click-through rates (CTR) [32,41]. The applied CTR estimation models today are mostly linear, ranging from logistic regression [32] and naive Bayes [14] to FTRL logistic regression [28] and Bayesian probit regression [12],

© Springer International Publishing Switzerland 2016
N. Ferro et al. (Eds.): ECIR 2016, LNCS 9626, pp. 45–57, 2016.
DOI: 10.1007/978-3-319-30671-1_4

all of which are based on a huge number of sparse features with one-hot encoding [1]. Linear models have advantages of easy implementation, efficient learning but relative low performance because of the failure of learning the non-trivial patterns to catch the interactions between the assumed (conditionally) independent raw features [12]. Non-linear models, on the other hand, are able to utilise different feature combinations and thus could potentially improve estimation performance. For example, factorisation machines (FMs) [29] map the user and item binary features into a low dimensional continuous space. And the feature interaction is automatically explored via vector inner product. Gradient boosting trees [38] automatically learn feature combinations while growing each decision/regression tree. However, these models cannot make use of all possible combinations of different features [20]. In addition, many models require feature engineering that manually designs what the inputs should be. Another problem of the mainstream ad CTR estimation models is that most prediction models have shallow structures and have limited expression to model the underlying patterns from complex and massive data [15]. As a result, their data modelling and generalisation ability is still restricted.

Deep learning [25] has become successful in computer vision [22], speech recognition [13], and natural language processing (NLP) [19,33] during recent five years. As visual, aural, and textual signals are known to be spatially and/or temporally correlated, the newly introduced unsupervised training on deep structures [18] would be able to explore such *local* dependency and establish a *dense* representation of the feature space, making neural network models effective in learning high-order features directly from the raw feature input. With such learning ability, deep learning would be a good candidate to estimate online user response rate such as ad CTR. However, most input features in CTR estimation are of multi-field and are discrete categorical features, e.g., the user location city (`London`, `Paris`), device type (`PC`, `Mobile`), ad category (`Sports`, `Electronics`) etc., and their local dependencies (thus the sparsity in the feature space) are unknown. Therefore, it is of great interest to see how deep learning improves the CTR estimation via learning feature representation on such large-scale multi-field discrete categorical features. To our best knowledge, there is no previous literature of ad CTR estimation using deep learning methods thus far[1]. In addition, training deep neural networks (DNNs) on a large input feature space requires tuning a huge number of parameters, which is computationally expensive. For instance, unlike image and audio cases, we have about 1 million binary input features and 100 hidden units in the first layer; then it requires 100 million links to build the first layer neural network.

In this paper, we take ad CTR estimation as a working example to study deep learning over a large multi-field categorical feature space by using embedding methods in both supervised and unsupervised fashions. We introduce two types of deep learning models, called Factorisation Machine supported Neural Network (FNN) and Sampling-based Neural Network (SNN). Specifically, FNN with

[1] Although the leverage of deep learning models on ad CTR estimation has been claimed in industry (e.g., [42]), there is no detail of the models or implementation.

a supervised-learning embedding layer using factorisation machines [31] is proposed to efficiently reduce the dimension from sparse features to dense continuous features. The second model SNN is a deep neural network powered by a sampling-based restricted Boltzmann machine (SNN-RBM) or a sampling-based denoising auto-encoder (SNN-DAE) with a proposed negative sampling method. Based on the embedding layer, we build multiple layers neural nets with full connections to explore non-trivial data patterns. Our experiments on multiple real-world advertisers' ad click data have demonstrated the consistent improvement of CTR estimation from our proposed models over the state-of-the-art ones.

2 Related Work

Click-through rate, defined as the probability of the ad click from a specific user on a displayed ad, is essential in online advertising [39]. In order to maximise revenue and user satisfaction, online advertising platforms must predict the expected user behaviour for each displayed ad and maximise the expectation that users will click. The majority of current models use logistic regression based on a set of sparse binary features converted from the original categorical features via one-hot encoding [26,32]. Heavy engineering efforts are needed to design features such as locations, top unigrams, combination features, etc. [15].

Embedding very large feature vector into low-dimensional vector spaces is useful for prediction task as it reduces the data and model complexity and improves both the effectiveness and the efficiency of the training and prediction. Various methods of embedding architectures have been proposed [23,37]. Factorisation machine (FM) [31], originally proposed for collaborative filtering recommendation, is regarded as one of the most successful embedding models. FM naturally has the capability of estimating interactions between any two features via mapping them into vectors in a low-rank latent space.

Deep Learning [2] is a branch of artificial intelligence research that attempts to develop the techniques that will allow computers to handle complex tasks such as recognition and prediction at high performance. Deep neural networks (DNNs) are able to extract the hidden structures and intrinsic patterns at different levels of abstractions from training data. DNNs have been successfully applied in computer vision [40], speech recognition [8] and natural language processing (NLP) [7,19,33]. Furthermore, with the help of unsupervised pre-training, we can get good feature representation which guides the learning towards basins of attraction of minima that support better generalisation from the training data [10]. Usually, these deep models have two stages in learning [18]: the first stage performs model initialisation via unsupervised learning (i.e., the restricted Boltzmann machine or stacked denoising auto-encoders) to make the model catch the input data distribution; the second stage involves a fine tuning of the initialised model via supervised learning with back-propagation. The novelty of our deep learning models lies in the first layer initialisation, where the input raw features are high dimensional and sparse binary features converted from the raw categorical features, which makes it hard to train traditional DNNs in large scale.

Compared with the word-embedding techniques used in NLP [19,33], our models deal with more general multi-field categorical features without any assumed data structures such as word alignment and letter-n-gram etc.

3 DNNs for CTR Estimation Given Categorical Features

In this section, we discuss the two proposed DNN architectures in detail, namely Factorisation-machine supported Neural Networks (FNN) and Sampling-based Neural Networks (SNN). The input categorical features are field-wise one-hot encoded. For each field, e.g., `city`, there are multiple units, each of which represents a specific value of this field, e.g., `city=London`, and there is only one positive (1) unit, while all others are negative (0). The encoded features, denoted as $\boldsymbol{x}$, are the input of many CTR estimation models [26,32] as well as our DNN models, as depicted at the bottom layer of Fig. 1.

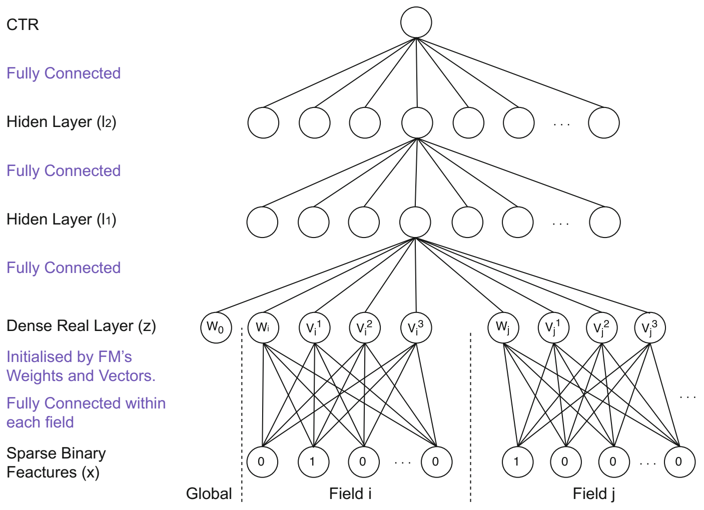


Fig. 1. A 4-layer FNN model structure.

3.1 Factorisation-Machine Supported Neural Networks (FNN)

Our first model FNN is based on the factorisation machine as the bottom layer. The network structure is shown in Fig. 1. With a top-down description, the output unit is a real number $\hat{y} \in (0,1)$ as predicted CTR, i.e., the probability of a specific user clicking a given ad in a certain context:

$$\hat{y} = \text{sigmoid}(\boldsymbol{W}_3 \boldsymbol{l}_2 + b_3), \tag{1}$$

where $\text{sigmoid}(x) = 1/(1+e^{-x})$ is the logistic activation function, $\boldsymbol{W}_3 \in \mathbb{R}^{1\times L}$, $b_3 \in \mathbb{R}$ and $\boldsymbol{l}_2 \in \mathbb{R}^L$ as input for this layer. The calculation of $\boldsymbol{l}_2$ is

$$\boldsymbol{l}_2 = \tanh(\boldsymbol{W}_2 \boldsymbol{l}_1 + \boldsymbol{b}_2), \tag{2}$$

where $\tanh(x) = (1 - e^{-2x})/(1 + e^{-2x})$, $\boldsymbol{W}_2 \in \mathbb{R}^{L \times M}$, $\boldsymbol{b}_2 \in \mathbb{R}^L$ and $\boldsymbol{l}_1 \in \mathbb{R}^M$. We choose $\tanh(\cdot)$ as it has optimal empirical learning performance than other activation functions, as will be discussed in Sect. 4.3. Similarly,

$$\boldsymbol{l}_1 = \tanh(\boldsymbol{W}_1 \boldsymbol{z} + \boldsymbol{b}_1), \tag{3}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{M \times J}$, $\boldsymbol{b}_1 \in \mathbb{R}^M$ and $\boldsymbol{z} \in \mathbb{R}^J$.

$$\boldsymbol{z} = (w_0, \boldsymbol{z}_1, \boldsymbol{z}_2, ...\boldsymbol{z}_i, ..., \boldsymbol{z}_n), \tag{4}$$

where $w_0 \in \mathbb{R}$ is a global scalar parameter and n is the number of fields in total. $\boldsymbol{z}_i \in \mathbb{R}^{K+1}$ is a parameter vectors for the i-th field in factorisation machines:

$$\boldsymbol{z}_i = \boldsymbol{W}_0^i \cdot \boldsymbol{x}[\text{start}_i : \text{end}_i] = (w_i, v_i^1, v_i^2, \ldots, v_i^K), \tag{5}$$

where start_i and end_i are starting and ending feature indexes of the i-th field, $\boldsymbol{W}_0^i \in \mathbb{R}^{(K+1)\times(\text{end}_i - \text{start}_i + 1)}$ and $\boldsymbol{x}$ is the input vector as described at beginning. All weights $\boldsymbol{W}_0^i$ are initialised with the bias term w_i and vector $\boldsymbol{v}_i$ respectively (e.g., $\boldsymbol{W}_0^i[0]$ is initialised by w_i, $\boldsymbol{W}_0^i[1]$ is initialised by v_i^1, $\boldsymbol{W}_0^i[2]$ is initialised by v_i^2, etc.). In this way, $\boldsymbol{z}$ vector of the first layer is initialised as shown in Fig. 1 via training a factorisation machine (FM) [31]:

$$y_{\text{FM}}(\boldsymbol{x}) := \text{sigmoid}\Big(w_0 + \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle x_i x_j\Big), \tag{6}$$

where each feature i is assigned with a bias weight w_i and a K-dimensional vector $\boldsymbol{v}_i$ and the feature interaction is modelled as their vectors' inner product $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$. In this way, the above neural nets can learn more efficiently from factorisation machine representation so that the computational complexity problem of the high-dimensional binary inputs has been naturally bypassed. Different hidden layers can be regarded as different internal functions capturing different forms of representations of the data instance. For this reason, this model has more abilities of catching intrinsic data patterns and leads to better performance.

The idea using FM in the bottom layer is ignited by Convolutional Neural Networks (CNNs) [11], which exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. Similarly, the inputs of hidden layer 1 are connected to the input units of a specific field. Also, the bottom layer is not fully connected as FM performs a field-wise training for one-hot sparse encoded input, allowing local sparsity, illustrated as the dash lines in Fig. 1. FM learns good structural data representation in the latent space, helpful for any further model to build on. A subtle difference, though, appears between the product rule of FM and the sum rule of DNN for combination. However, according to [21], if the observational discriminatory information is highly ambiguous (which is true in our case for ad click behaviour), the posterior weights (from DNN) will not deviate dramatically from the prior (FM).

Furthermore, the weights in hidden layers (except the FM layer) are initialised by layer-wise RBM pre-training [3] using contrastive divergence [17], which effectively preserves the information in input dataset as detailed in [16,18]. The initial weights for FMs are trained by stochastic gradient descent (SGD), as detailed in [31]. Note that we only need to update weights which connect to the positive input units, which largely reduces the computational complexity. After pre-training of the FM and upper layers, supervised fine-tuning (back propagation) is applied to minimise loss function of cross entropy:

$$L(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}), \tag{7}$$

where $\hat{y}$ is the predicted CTR in Eq. (1) and y is the binary click ground-truth label. Using the chain rule of back propagation, the FNN weights including FM weights can be efficiently updated. For example, we update FM layer weights via

$$\frac{\partial L(y,\hat{y})}{\partial \boldsymbol{W}_0^i} = \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}_i} \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{W}_0^i} = \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}_i} \boldsymbol{x}[\text{start}_i : \text{end}_i] \tag{8}$$

$$\boldsymbol{W}_0^i \leftarrow \boldsymbol{W}_0^i - \eta \cdot \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}_i} \boldsymbol{x}[\text{start}_i : \text{end}_i]. \tag{9}$$

Due to the fact that the majority entries of $\boldsymbol{x}[\text{start}_i : \text{end}_i]$ are 0, we can accelerate fine-tuning by updating weights linking to positive units only.

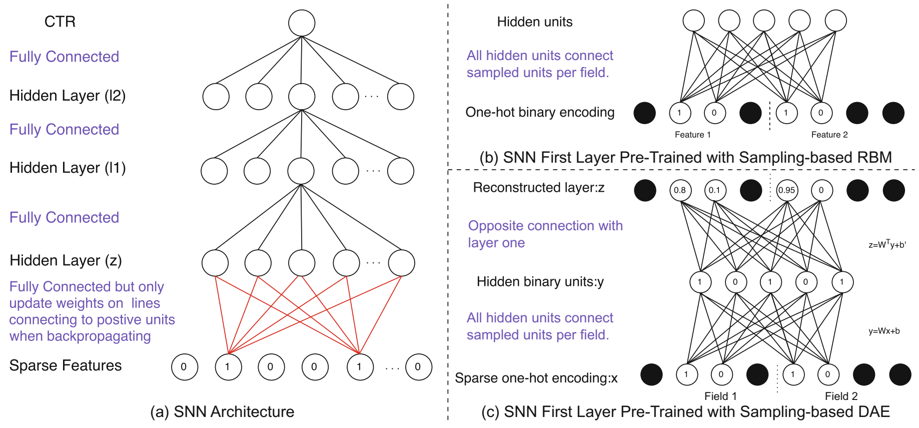

CTR
Fully Connected
Hidden Layer (l2)
Fully Connected
Hidden Layer (l1)
Fully Connected
Hidden Layer (z)
Fully Connected but only update weights on lines connecting to postive units when backpropagating
Sparse Features
(a) SNN Architecture
Hidden units
All hidden units connect sampled units per field.
One-hot binary encoding
Feature 1
Feature 2
(b) SNN First Layer Pre-Trained with Sampling-based RBM
Reconstructed layer:z
0.8
0.1
0.95
Opposite connection with layer one
z=W^Ty+b'
Hidden binary units:y
All hidden units connect sampled units per field.
y=Wx+b
Sparse one-hot encoding:x
Field 1
Field 2
(c) SNN First Layer Pre-Trained with Sampling-based DAE

Fig. 2. A 4-layer SNN architecture and two first-layer pre-training methods.

3.2 Sampling-Based Neural Networks (SNN)

The structure of the second model SNN is shown in Fig. 2(a). The difference between SNN and FNN lies in the structure and training method in the bottom layer. SNN's bottom layer is fully connected with sigmoid activation function:

$$\boldsymbol{z} = \text{sigmoid}(\boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{b}_0). \tag{10}$$

To initialise the weights of the bottom layer, we tried both restricted Boltzmann machine (RBM) [16] and denoising auto-encoder (DAE) [4] in the pre-training stage. In order to deal with the computational problem of training large sparse one-hot encoding data, we propose a sampling-based RBM (Fig. 2(b), denoted as SNN-RBM) and a sampling-based DAE in (Fig. 2(c), denoted as SNN-DAE) to efficiently calculate the initial weights of the bottom layer.

Instead of modelling the whole feature set for each training instance set, for each feature field, e.g., `city`, there is only one positive value feature for each training instance, e.g., `city=London`, we sample m negative units, e.g., `city=Paris` when $m = 1$, randomly with value 0. Black units in Fig. 2(b) and (c) are unsampled and thus ignored when pre-training the data instance. With the sampled units, we can train an RBM via contrastive divergence [17] and a DAE via SGD with unsupervised approaches to largely reduce the data dimension with high recovery performance. The real-value dense vector is used as the input of the further layers in SNN.

In this way, computational complexity can be dramatically reduced and, in turn, initial weights can be calculated quickly and back-propagation is then performed to fine-tune SNN model.

3.3 Regularisation

To prevent overfitting, the widely used L2 regularisation term is added to the loss function. For example, the L2 regularisation for FNN in Fig. 1 is

$$\Omega(\boldsymbol{w}) = ||\boldsymbol{W}_0||_2^2 + \sum_{l=1}^{3} \Big(||\boldsymbol{W}_l||_2^2 + ||\boldsymbol{b}_l||_2^2 \Big). \tag{11}$$

On the other hand, *dropout* [35] is a technique which becomes a popular and effective regularisation technique for deep learning during the recent years. We also implement this regularisation and compare them in our experiment.

4 Experiment

4.1 Experiment Setup

Data. We evaluate our models based on iPinYou dataset [27], a public real-world display ad dataset with each ad display information and corresponding user click feedback. The data logs are organised by different advertisers and in a row-per-record format. There are 19.50 M data instances with 14.79 K positive label (click) in total. The features for each data instance are all categorical. Feature examples in the ad log data are `user agent`, partially masked `IP`, `region`, `city`, `ad exchange`, `domain`, `URL`, `ad slot ID`, `ad slot visibility`, `ad slot size`, `ad slot format`, `creative ID`, `user tags`, etc. After one-hot encoding, the number of binary features is 937.67 K in the whole dataset. We feed each compared model with these binary-feature data instances and the user click (1) and non-click (0) feedback as the ground-truth labels. In our experiments, we use training data from advertiser 1458, 2259, 2261, 2997, 3386 and the whole dataset, respectively.

Models. We compare the performance of the following CTR estimation models:

LR: Logistic Regression [32] is a linear model with simple implementation and fast training speed, which is widely used in online advertising estimation.

FM: Factorisation Machine [31] is a non-linear model able to estimate feature interactions even in problems with huge sparsity.

FNN: Factorisation-machine supported Neural Network is our proposed model as described in Sect. 3.1.

SNN: Sampling-based Neural Network is also our proposed model with sampling-based RBM and DAE pre-training methods for the first layer in Sect. 3.2, denoted as SNN-RBM and SNN-DAE respectively.

Our experiment code[2] of both FNN and SNN is implemented with `Theano`[3].

Metric. To measure the CTR estimation performance of each model, we employ the area under ROC curve (AUC)[4]. The AUC [12] metric is a widely used measure for evaluating the CTR performance.

4.2 Performance Comparison

Table 1 shows the results that compare LR, FM, FNN and SNN with RBM and DAE on 5 different advertisers and the whole dataset. We observe that FM is not significantly better than LR, which means 2-order combination features might not be good enough to catch the underlying data patterns. The AUC performance of the proposed FNN and SNN is better than the performance of

Table 1. Overall CTR estimation AUC performance.

	LR	FM	FNN	SNN-DAE	SNN-RBM
1458	70.42 %	70.21 %	**70.52 %**	70.46 %	70.49 %
2259	69.66 %	69.73 %	**69.74 %**	68.08 %	68.34 %
2261	62.03 %	60.97 %	62.99 %	**63.72 %**	**63.72 %**
2997	60.77 %	60.87 %	61.41 %	**61.58 %**	61.45 %
3386	80.30 %	79.05 %	**80.56 %**	79.62 %	80.07 %
all	68.81 %	68.18 %	**70.70 %**	69.15 %	69.15 %

[2] The source code with demo data: https://github.com/wnzhang/deep-ctr.

[3] `Theano`: http://deeplearning.net/software/theano/.

[4] Besides AUC, root mean square error (RMSE) is also tested. However, positive/negative examples are largly unbalanced in ad click scenario, and the empirically best regression model usually provides the predicted CTR close to 0, which results in very small RMSE values and thus the improvement is not well captured.

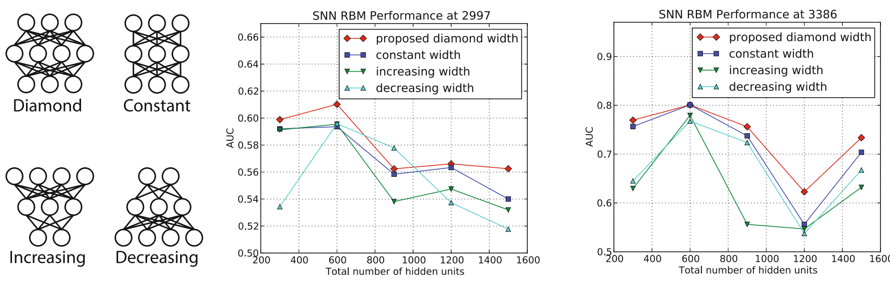
LR and FM on all tested datasets. Based on the latent structure learned by FM, FNN further learns effective patterns between these latent features and provides a consistent improvement over FM. The performance of SNN-DAE and SNN-RBM is generally consistent, i.e., the relative order of the results of the SNN are almost the same.

4.3 Hyperparameter Tuning

Due to the fact that deep neural networks involve many implementation details and need to tune a fairly large number of hyper-parameters, following details show how we implement our models and tune hyperparameters in the models.

We use stochastic gradient descent to learn most of our parameters for all proposed models. Regarding selecting the number of training epochs, we use early stopping [30], i.e., the training stops when the validation error increases. We try different learning rate from 1, 0.1, 0.01, 0.001 to 0.0001 and choose the one with optimal performance on the validation dataset.

For negative unit sampling of SNN-RBM and SNN-DAE, we try the negative sample number $m = 1, 2$ and 4 per field as described in Sect. 3.2, and find $m = 2$ produces the best results in most situations. For the activation functions in both models on the hidden layers (as Eqs. (3) and (2)), we try linear function, sigmoid function and tanh function, and find the result of tanh function is optimal. This might be because the hyperbolic tangent often converges faster than the sigmoid function.



(a) Architecture (b) SNN-RBM on 2997 (c) SNN-RBM on 3386

Fig. 3. AUC Performance with different architectures.

4.4 Architecture Selection

In our models, we investigate architectures with 3, 4 and 5 hidden layers by fixing all layer sizes and find the architecture with 3 hidden layers (i.e., 5 layers in total) is the best in terms of AUC performance. However, the range of choosing their layer sizes is exponential in the number of hidden layers. Suppose there is a deep neural network with L hidden layers and each of the hidden layers is trained with a range of hidden units from 100 to 500 with increments of 100, thus there are 5^L models in total to compare.

Instead of trying all combinations of hidden units, in our experiment we use another strategy by starting tuning the different hidden layer sizes with the same number of hidden units in all three hidden layers[5] since the architecture with equal-size hidden layers is empirically better than the architecture with increasing width or decreasing width in [24]. For this reason, we start tuning layer sizes with equal hidden layer sizes. In fact, apart from increasing, constant, decreasing layer sizes, there is a more effective structure, which is the diamond shape of neural networks, as shown in Fig. 3(a). We compare our diamond shape network with other three shapes of networks and tune the total number of total hidden units on two different datasets shown in Fig. 3(b) and (c). The diamond shape architecture outperforms others in almost all layer size settings. The reason why this diamond shape works might be because this special shape of neural network has certain constraint to the capacity of the neural network, which provides better generalisation on test sets. On the other hand, the performance of diamond architecture picks at the total hidden unit size of 600, i.e., the combination of (200, 300, 100). This depends on the training data observation numbers. Too many hidden units against a limited dataset could cause overfitting.

4.5 Regularisation Comparison

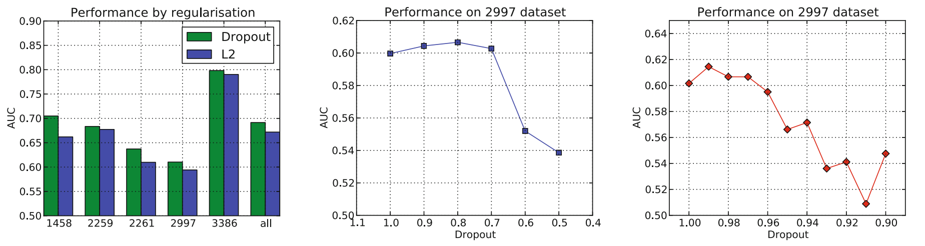
Neural network training algorithms are very sensitive to the overfitting problem since deep networks have multiple non-linear layers, which makes them very expressive models that can learn very complicated functions. For DNN models, we compared L2 regularisation (Eq. (11)) and dropout [35] for preventing complex co-adaptations on the training data. The dropout rate implemented in this experiment refers to the probability of each unit being active.

Figure 4(a) shows the compared AUC performance of SNN-RBM regularised by L2 norm and dropout. It is obvious that dropout outperforms L2 in all compared settings. The reason why dropout is more effective is that when feeding each training case, each hidden unit is stochastically excluded from the network with a probability of dropout rate, i.e., each training case can be regarded as a new model and these models are averaged as a special case of bagging [5], which effectively improves the generalisation ability of DNN models.

4.6 Analysis of Parameters

As a summary of Sects. 4.4 and 4.5, for both FNN and SNN, there are two important parameters which should be tuned to make the model more effective: (i) the parameters of layer size decide the architecture of the neural network and (ii) the parameter of dropout rate changes generalisation ability on all datasets compared to neural networks just with L2 regularisation.

[5] Some advanced Bayesian methods for hyperparameter tuning [34] are not considered in this paper and may be investigated in the future work.



(a) Dropout vs. L2 (b) FNN on 2997 dataset (c) SNN on 2997 dataset

Fig. 4. AUC performance w.r.t difference regularisation settings.

Figure 4(b) and (c) show how the AUC performance changes with the increasing of dropout in both FNN and SNN. We can find that there is an upward trend of performance in both models at the beginning and then drop sharply with continuous decreasing of dropout rate. The distinction between two models is the different sensitivities of the dropout. From Fig. 4(c), we can see the model SNN is sensitive to the dropout rate. This might be caused by the connectivities in the bottom layer. The bottom layer of the SNN is fully connected with the input vector while the bottom layer for FNN is partially connected and thus the FNN is more robust when some hidden units are dropped out. Furthermore, the sigmoid activation function tend to more effective than the linear activation function in terms of dropout. Therefore, the dropout rates at the best performance of FNN and SNN are quite different. For FNN the optimal dropout rate is around 0.8 while for SNN is about 0.99.

5 Conclusion

In this paper, we investigated the potential of training deep neural networks (DNNs) to predict users' ad click response based on multi-field categorical features. To deal with the computational complexity problem of high-dimensional discrete categorical features, we proposed two DNN models: field-wise feature embedding with supervised factorisation machine pre-training, and fully connected DNN with field-wise sampling-based RBM and DAE unsupervised pre-training. These architectures and pre-training algorithms make our DNNs trained very efficiently. Comprehensive experiments on a public real-world dataset verifies that the proposed DNN models successfully learn the underlying data patterns and provide superior CTR estimation performance than other compared models. The proposed models are very general and could enable a wide range of future works. For example, the model performance can be improved by momentum methods in that it suffices for handling the curvature problems in DNN training objectives without using complex second-order methods [36]. In addition, the partial connection in the bottom layer could be extended to higher hidden layers as partial connectivities have many advantages such as lower complexity, higher generalisation ability and more similar to human brain [9].

References

1. Beck, J.E., Park Woolf, B.: High-level student modeling with machine learning. In: Gauthier, G., VanLehn, K., Frasson, C. (eds.) ITS 2000. LNCS, vol. 1839, pp. 584–593. Springer, Heidelberg (2000)
2. Bengio, Y.: Learning deep architectures for AI. Found. Trends Mach. Learn. **2**(1), 1–127 (2009)
3. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al.: Greedy layer-wise training of deep networks. In: NIPS, vol. 19, p. 153 (2007)
4. Bengio, Y., Yao, L., Alain, G., Vincent, P.: Generalized denoising auto-encoders as generative models. In: NIPS, pp. 899–907 (2013)
5. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
6. Broder, A.Z.: Computational advertising. In: SODA, vol. 8, pp. 992–992 (2008)
7. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. JMLR **12**, 2493–2537 (2011)
8. Deng, L., Abdel-Hamid, O., Yu, D.: A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In: ICASSP, pp. 6669–6673. IEEE (2013)
9. Elizondo, D., Fiesler, E.: A survey of partially connected neural networks. Int. J. Neural Syst. **8**(05n06), 535–558 (1997)
10. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? JMLR **11**, 625–660 (2010)
11. Fukushima, K.: Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybern. **36**(4), 193–202 (1980)
12. Graepel, T., Candela, J.Q., Borchert, T., Herbrich, R.: Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In: ICML, pp. 13–20 (2010)
13. Graves, A., Mohamed, A., Hinton, G.: Speech recognition with deep recurrent neural networks. In: ICASSP, pp. 6645–6649. IEEE (2013)
14. Hand, D.J., Yu, K.: Idiot's bayes not so stupid after all? Int. Statist. Rev. **69**(3), 385–398 (2001)
15. He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al.: Practical lessons from predicting clicks on ads at facebook. In: ADKDD, pp. 1–9. ACM (2014)
16. Hinton, G.: A practical guide to training restricted boltzmann machines. Momentum **9**(1), 926 (2010)
17. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural comput. **14**(8), 1771–1800 (2002)
18. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
19. Huang, P.S., He, X., Gao, J., Deng, L., Acero, A., Heck, L.: Learning deep structured semantic models for web search using clickthrough data. In: CIKM, pp. 2333–2338 (2013)
20. Juan, Y.C., Zhuang, Y., Chin, W.S.: 3 idiots approach for display advertising challenge. In: Internet and Network Economics, pp. 254–265. Springer, Heidelberg (2011)
21. Kittler, J., Hatef, M., Duin, R.P., Matas, J.: On combining classifiers. PAMI **20**(3), 226–239 (1998)

22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)
23. Kurashima, T., Iwata, T., Takaya, N., Sawada, H.: Probabilistic latent network visualization: inferring and embedding diffusion networks. In: KDD, pp. 1236–1245. ACM (2014)
24. Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P.: Exploring strategies for training deep neural networks. JMLR **10**, 1–40 (2009)
25. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553) (2015)
26. Lee, K., Orten, B., Dasdan, A., Li, W.: Estimating conversion rate in display advertising from past performance data. In: KDD, pp. 768–776. ACM (2012)
27. Liao, H., Peng, L., Liu, Z., Shen, X.: ipinyou global rtb bidding algorithm competition dataset. In: ADKDD, pp. 1–6. ACM (2014)
28. McMahan, H.B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al.: Ad click prediction: a view from the trenches. In: KDD, pp. 1222–1230. ACM (2013)
29. Oentaryo, R.J., Lim, E.P., Low, D.J.W., Lo, D., Finegold, M.: Predicting response in mobile advertising with hierarchical importance-aware factorization machine. In: WSDM (2014)
30. Prechelt, L.: Automatic early stopping using cross validation: quantifying the criteria. Neural Netw. **11**(4), 761–767 (1998)
31. Rendle, S.: Factorization machines with libfm. ACM TIST **3**(3), 57 (2012)
32. Richardson, M., Dominowska, E., Ragno, R.: Predicting clicks: estimating the click-through rate for new ads. In: WWW, pp. 521–530. ACM (2007)
33. Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G.: A latent semantic model with convolutional-pooling structure for information retrieval. In: CIKM (2014)
34. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: NIPS, pp. 2951–2959 (2012)
35. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. JMLR **15**(1), 1929–1958 (2014)
36. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: ICML, pp. 1139–1147 (2013)
37. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: WWW, pp. 1067–1077 (2015)
38. Trofimov, I., Kornetova, A., Topinskiy, V.: Using boosted trees for click-through rate prediction for sponsored search. In: WINE, p. 2. ACM (2012)
39. Wang, X., Li, W., Cui, Y., Zhang, R., Mao, J.: Click-through rate estimation for rare events in online advertising. In: Online Multimedia Advertising: Techniques and Technologies, pp. 1–12 (2010)
40. Zeiler, M.D., Taylor, G.W., Fergus, R.: Adaptive deconvolutional networks for mid and high level feature learning. In: ICCV, pp. 2018–2025. IEEE (2011)
41. Zhang, W., Yuan, S., Wang, J.: Optimal real-time bidding for display advertising. In: KDD, pp. 1077–1086. ACM (2014)
42. Zou, Y., Jin, X., Li, Y., Guo, Z., Wang, E., Xiao, B.: Mariana: Tencent deep learning platform and its applications. VLDB **7**(13), 1772–1777 (2014)