Entity Embeddings of Categorical Variables

Cheng Guo* and Felix Berkhahn†
Neokami Inc.
(Dated: April 25, 2016)

We map categorical variables in a function approximation problem into Euclidean spaces, which are the entity embeddings of the categorical variables. The mapping is learned by a neural network during the standard supervised training process. Entity embedding not only reduces memory usage and speeds up neural networks compared with one-hot encoding, but more importantly by mapping similar values close to each other in the embedding space it reveals the intrinsic properties of the categorical variables. We applied it successfully in a recent Kaggle competition[a] and were able to reach the third position with relative simple features. We further demonstrate in this paper that entity embedding helps the neural network to generalize better when the data is sparse and statistics is unknown. Thus it is especially useful for datasets with lots of high cardinality features, where other methods tend to overfit. We also demonstrate that the embeddings obtained from the trained neural network boost the performance of all tested machine learning methods considerably when used as the input features instead. As entity embedding defines a distance measure for categorical variables it can be used for visualizing categorical data and for data clustering.

I. INTRODUCTION

Many advances have been achieved in the past 15 years in the field of neural networks due to a combination of faster computers, more data and better methods [1]. Neural networks revolutionized computer vision[2–6], speech recognition[7, 8] and natural language processing[9–12] and have replaced or are replacing the long dominating methods in each field.

Unlike in the above fields where data is unstructured, neural networks are not as prominent when dealing with machine learning problems with structured data. This can be easily seen by the fact that the top teams in many online machine learning competitions like those hosted on Kaggle use tree based methods more often than neural networks[13].

To understand this, we compared neural network and decision tree's approach to the general machine learning problem, which is to approximate the function

$$y = f(x_1, x_2, ..., x_n). \tag{1}$$

Given a set of input values $(x_1, x_2, ..., x_n)$ it generates the target output value y.

In principle a neural network can approximate any continuous function[14, 15] and piece wise continuous function [16]. However, it is not suitable to approximate arbitrary non-continuous functions as it assumes certain level of continuity in its general form. During the training phase the continuity of the data guarantees the convergence of the optimization, and during the prediction phase it ensures that slightly changing the values of the input keeps the output stable. On the other hand decision trees do not assume any continuity of the feature variables and can divide the states of a variable as fine as necessary.

Interestingly the problems we usually face in nature are often continuous if we use the right representation of data. Whenever we find a better way to reveal the continuity of the data we increase the power of neural networks to learn the data. For example, convolutional neural networks [17] group pixels in the same neighborhood together. This increases the continuity of the data compared to simply representing the image as a flattened vector of all the pixel values of the images. The rise of neural networks in natural language processing is based on the word embedding [9, 11, 18] which puts words with similar meaning closer to each other in a word space thus increasing the continuity of the words compared to using one-hot encoding of words.

Unlike unstructured data found in nature, structured data with categorical features may not have continuity at all and even if it has it may not be so obvious. The continuous nature of neural networks limits their applicability to categorical variables. Therefore, naively applying neural networks on structured data with integer representation for category variables does not work well. A common way to circumvent this problem is to use one-hot encoding, but it has two shortcomings: First when we have many high cardinality features one-hot encoding often results in an unrealistic amount of computational resource requirement. Second, it treats different values of categorical variables completely independent of each other and often ignores the informative relations between them.

In this paper we show how to use the entity embedding method to automatically learn the representation of categorical features in multi-dimensional spaces which puts values with similar effect in the function approximation problem Eq. (1) close to each other, and thereby reveals the intrinsic continuity of the data and helps neural networks as well as other common machine learning algorithms to solve the problem.

* cheng.guo.work@gmail.com
† felix.berkhahn@gmail.com
[a] `https://www.kaggle.com/c/rossmann-store-sales`

Distributed representation of entities has been used in many contexts before[19–21]. Our main contributions are: First we explored this idea in the general function approximation problem and demonstrated its power in a large machine learning competition. Second we studied the properties of the learned embeddings and showed how the embeddings can be used to understand and visualize categorical data.

II. RELATED WORK

As far as we know the first domain where the entity embedding method in the context of neural networks has been explored is the representation of relational data[19]. More recently, knowledge base which is a large collection of complex relational data is seeing lots of works using entity embedding[22–24]. The basic data structure of relational data is triplets (h, r, t), where h and t are two entities and r is the relation. The entities are mapped to vectors and relations are sometimes mapped to a matrix(e.g. Linear Relation Embedding [25]) or two matrices(e.g. Structured Embeddings[26]) or a vector in the same embedding space as the entities[27] etc. Various kind of score function can be defined (see Table. 1 of [28]) to measure the likelihood of such a triplet, and the score function is used as the objective function for learning the embeddings.

In natural language processing, Word embeddings have been used to map words and phrases [9] into a continuous distributed vector in a semantic space. In this space similar words are closer. What is even more interesting is that not only the distance between words are meaningful but also the direction of the difference vectors. For example, it has been observed [11] that the learned word vectors have relations such as:

$$\mathbf{King} - \mathbf{Man} \approx \mathbf{Queen} - \mathbf{Woman} \tag{2}$$

$$\mathbf{Paris} - \mathbf{France} \approx \mathbf{Rome} - \mathbf{Italy} \tag{3}$$

There are many ways [9, 11, 18, 29, 30] to learn word embeddings. A very fast way [31] is to use the word context with the aim to maximize

$$p(w_c|w) = \frac{\exp(\mathbf{w} \cdot \mathbf{w_c})}{\sum_i \exp(\mathbf{w} \cdot \mathbf{w_i})}, \tag{4}$$

where $\mathbf{w}$ and $\mathbf{w_c}$ are the vector representation of a word w and its neighbor word w_c inside the context window while $p(w_c|w)$ is the probability to have w_c in the context of w. The sum is over the whole vocabulary. Word embeddings can also be learned with supervised methods. For example in Ref. [30] the embeddings can be learned using text with labeled sentiment. This approach is very close to the approach we use in this paper but in a different context.

III. TREE BASED METHODS

As tree based methods are the most widely used method for structured data and they are the main methods we are comparing to, we will briefly review them here. Random Forests and in particular Gradient Boosted Trees have proven their capabilities in numerous recent Kaggle competitions [13]. In the following, we will briefly describe the process of growing a single decision tree used for regression, as well as two popular tree ensemble methods: random forests and gradient tree boosting.

A. Single decision tree

Decision trees partition the feature space X into M different sub-spaces $R_1, R_2, \ldots R_M$. The function f in equation (1) is thus modeled as

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{5}$$

with I being the indicator function $I(x \in R_m) = \begin{cases} 1 & \text{if } x \in R_m \\ 0 & \text{else} \end{cases}$. Using the common sum of squares

$$L = \sum_i (y_i - f(x_i))^2 \tag{6}$$

as loss function, it follows from standard linear regression theory that, for given R_m, the optimal choices for the parameters c_m are just the averages

$$\hat{c}_m = \frac{1}{|R_m|} \sum_{x_i \in R_m} y_i \tag{7}$$

with $|R_m|$ the number of elements in the set R_m. Ideally, we would try to find the optimal partition $\{R_m\}$ such as to minimize the loss function (6). However, this is not computationally feasible, as the number of possible partitions grows exponentially with the size of the feature space X. Instead, a greedy algorithm is applied, that tries to find subsequent splits of X that try to minimize (6) locally at each split. To start with, given a splitting variable j and a split point s, we define the pair of half-planes

$$R_1(j, s) = \{X | X_j \leq s\} \tag{8}$$

$$R_2(j, s) = \{X | X_j > s\} \tag{9}$$

and optimize (6) for j and s:

$$\min_{j,s} \left[\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2 \right] \tag{10}$$

The optimal choices for the parameters $\hat{c}_1$ and $\hat{c}_2$ follow directly from (7).

After (10) is solved for j and s, the same algorithm is applied recursively on the two half-planes R_1 and R_2 until the tree is fully grown.

The size up to which the tree is grown governs the complexity of the model and thus implies a bias-variance tradeoff: A very large tree likely overfits the training data, while a very small tree likely is not complex enough to capture the important dependencies in the data. There are several strategies and measures available to control the tree size. A very popular strategy is *pruning*, where first large trees are grown until they reach a minimal tree size (like minimum number of nodes or minimal height), and then internal nodes are collapsed (i.e. pruned) to minimize a cost-complexity measure C_α such as

$$C_\alpha = \sum_i (y_i - f(x_i))^2 + \alpha |T| \tag{11}$$

where $|T|$ is the number of terminal nodes in the tree T and α is a free parameter to control the complexity of the model.

B. Random forests

A single decision tree is a highly non-linear classifier with typically low bias but high variance. Random forests address the problem of high variance by establishing a committee (i.e. average) of identically distributed single decision trees.

To be precise, random forests contain N single decision trees grown by the following algorithm:

1. Draw a bootstrap sample from the training data, that is, select n random records from the training data.
2. Grow a single decision tree T_i as described in section III A, with the only difference that at each split-node m features are randomly picked that are considered for the best split at the split-node.
3. Output the ensemble of all decision trees $\{T_i\}_{i=1\ldots N}$.

For regression, an unseen sample is then predicted as:

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} T_i(x) \tag{12}$$

As all T_i are identically distributed, the linear average of (12) preserves the presumably low bias of a single decision tree. However, averaging will reduce the variance of the single decision trees.

C. Gradient boosted trees

Gradient tree boosting is another ensemble tree based method, that is we try to approximate $f(x)$ by a sum of trees T_i:

$$f(x) = \sum_{k=1}^{N} T_k(x) \tag{13}$$

For a generic loss function L (not necessarily quadratic), the n-th tree is grown on the quantity r_{in}

$$r_{in} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f=f_{n-1}} \tag{14}$$

computed using its $n-1$ predecessor trees. Here, the y_i are the target labels, x_i are the sample features and f_{n-1} is the sum of the first $n-1$ trees

$$f_{n-1}(x) = \sum_{k=1}^{n-1} T_k(x) \tag{15}$$

In case of a squared error loss $L = \sum_i (y_i - f(x_i))^2$ this amounts to fitting the n-th tree on the residuals $y_i - f_{n-1}(x_i)$ of its $n-1$ predecessor trees. Hence, equation (14) generalizes to a generic loss function by minimizing the loss function L iteratively at each step along the gradient descent direction in the space spanned by all possible trees T_n. This is where the name *gradient* boosted trees comes from.

As for every boosting algorithm, the next iterative classifier T_n tries to correct its T_{n-1} predecessors. Hence, in contrast to random forests, gradient tree boosting also aims to minimize the bias of the ensemble and not only the variance.

IV. STRUCTURED DATA

By structured data we mean data collected and organized in a table format with columns representing different features (variables) or target values and rows representing different samples. We focus on this type of data in this paper.

The most common variable types in structured data are continuous variables and discrete variables. Continuous variables such as temperature, price, weight can be represented by real numbers. Discrete variables such as age, color, bus line number can be represented by integers. Often the integers are just used for convenience to label the different states and have no information in themselves. For example if we use 1, 2, 3 to represent red, blue and yellow, one can not assume that "blue is bigger than red" or "the average of red and yellow are blue" or anything that introduces additional information based on the properties of integers. These integers are called nominal numbers. Other times there is an intrinsic ordering in the integer index such as age or month of the year. These integers are called cardinal number or ordinal numbers. Note that the meaning or order may not be more useful for the problem than only considering the

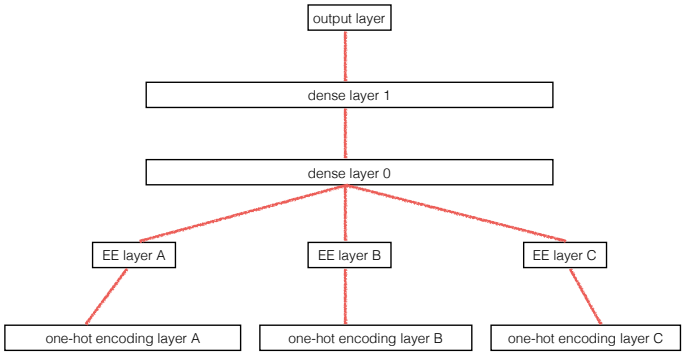

FIG. 1. Illustration that entity embedding layers are equivalent to extra layers on top of each one-hot encoded input.

integer as nominal numbers. For example the month ordering has nothing to do with number of days in a month (January is closer to Jun than February regarding number of days it has). Therefore we will treat both types of discrete variables in the same way. The task of entity embedding is to map discrete values to a multi-dimensional space where values with similar function output are close to each other.

V. ENTITY EMBEDDING

To learn the approximation of the function Eq. (1) we map each state of a discrete variable to a vector as

$$e_i : x_i \mapsto \mathbf{x}_i \tag{16}$$

This mapping is equivalent to an extra layer of linear neurons on top of the one-hot encoded input as shown in Fig. 1. To show this we represent one-hot encoding of x_i as

$$u_i : x_i \mapsto \delta_{x_i\alpha}, \tag{17}$$

where $\delta_{x_i\alpha}$ is Kronecker delta and the possible values for α are the same as x_i. If m_i is the number of values for the categorical variable x_i, then $\delta_{x_i\alpha}$ is a vector of length m_i, where the element is only non-zero when $\alpha = x_i$.

The output of the extra layer of linear neurons given the input x_i is defined as

$$\mathbf{x}_i \equiv \sum_\alpha w_{\alpha\beta}\delta_{x_i\alpha} = w_{x_i\beta} \tag{18}$$

where $w_{\alpha\beta}$ is the weight connecting the one-hot encoding layer to the embedding layer and β is the index of the embedding layer. Now we can see that the mapped embeddings are just the weights of this layer and can be learned in the same way as the parameters of other neural network layers.

After we use entity embeddings to represent all categorical variables, all embedding layers and the input of all continuous variables (if any) are concatenated. The merged layer is treated like a normal input layer in neural networks and other layers can be build on top of it. The whole network can be trained with the standard back-propagation method. In this way, the entity embedding layer learns about the intrinsic properties of each category, while the deeper layers form complex combinations of them.

The dimensions of the embedding layers D_i are hyper-parameters that need to be pre-defined. The bound of the dimensions of entity embeddings are between 1 and $m_i - 1$ where m_i is the number of values for the categorical variable x_i. In practice we chose the dimensions based on experiments. The following empirical guidelines are used during this process: First, the more complex the more dimensions. We roughly estimated how many features/aspects one might need to describe the entities and used that as the dimension to start with. Second, if we had no clue about the first guideline, then we started with $m_i - 1$.

It would be good to have more theoretical guidelines on how to choose D_i. We think this probably relates to the problem of embedding of finite metric space, and that is what we want to explore next.

A. Relation with embedding of finite metric space

With entity embedding we want to put similar values of a categorical variable closer to each other in the embedding space. If we use a real number to define similarity of the values then entity embedding is closely related to the embedding of finite metric space problem in topology.

We define a finite metric space (M_i, d_i) associated with each categorical variable x_i in the function approximation problem Eq. (1), where M_i is the set of all possible values of x_i. d_i is the metric on M_i, which is the distance function between any two pairs of values (x_i^p, x_i^q) of x_i. We want d_i to represent the similarity of (x_i^p, x_i^q). There are many ways to define it, one simple and natural way is

$$d_i(x_i^p, x_i^q) = \langle |f(x_i^p, \bar{\mathbf{x}}_\mathbf{i}) - f(x_i^q, \bar{\mathbf{x}}_\mathbf{i})| \rangle_{\bar{\mathbf{x}}_\mathbf{i}} \tag{19}$$

where $\langle \dots \rangle_{\bar{\mathbf{x}}_\mathbf{i}}$ is the average over all values of the parameters of f other than x_i. $\bar{\mathbf{x}}_\mathbf{i}$ is shorter notation for $(x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots)$. It can be verified that the following conditions hold for the metric Eq. (19):

$$d_i(x_i^p, x_i^q) = 0 \Leftrightarrow x_i^p = x_i^q \tag{20}$$
$$d_i(x_i^p, x_i^q) = d_i(x_i^q, x_i^p) \tag{21}$$
$$d_i(x_i^p, x_i^r) \leq d_i(x_i^p, x_i^q) + d_i(x_i^q, x_i^r) \tag{22}$$

Eq. (20) may not automatically hold in a real problem when two different values always generate the same output. However, this also means one value is redundant, and it is easy to simply merge these two values into one by redefining the categorical variable to make Eq. (20) hold.

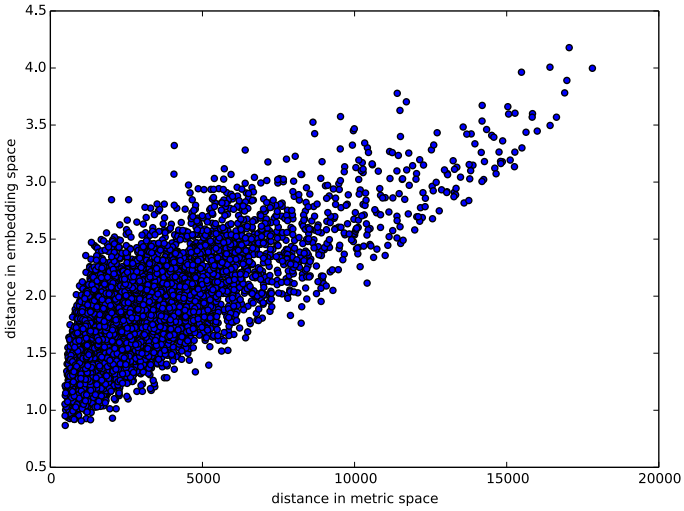

FIG. 2. Distance in the store embedding space versus distance in the metric space for 10000 random pair of stores.

Ref. [32] proved sufficient and necessary conditions to isometrically embed a generic metric space in an euclidean metric space. Applied on the metric Eq. (19), it would require that the matrix

$$(\boldsymbol{M_i})_{pq} = e^{-\lambda\langle|f(x_i^p,\bar{\mathbf{x}}_\mathbf{i})-f(x_i^q,\bar{\mathbf{x}}_\mathbf{i})|\rangle_{\bar{\mathbf{x}}_\mathbf{i}}} \tag{23}$$

is positive definite. We took the store feature (see Table I) as an example and verified this numerically and found that it is not true. Therefore the store metric space as we defined cannot be isometrically embedded in an Euclidean space.

What is the relation of the learned embeddings of a categorical variable to this metric space? To answer this question we plot in Fig. 2 the distance between 10000 random store pairs in the learned store embedding space and in the metric space as defined in Eq. (19). It is not an isometric embedding obviously. We can also see from the figure that there is a linear relation with well defined upper and lower boundary. Why are there clear boundaries and what does the shape mean? Is this related to some theorems regarding the distorted mapping of metric space[33, 34]? How is the distortion related to the embedding dimension D_i? If we apply multidimensional scaling[35] directly on the metric d_i how is the result different to the learned entity embeddings of the neural network? Due to time limit we will leave these interesting questions for future investigations.

VI. EXPERIMENTS

In this paper we will use the dataset from the Kaggle Rossmann Sale Prediction competition as an example. The goal of the competition is to predict the daily sales of each store of Dirk Rossmann GmbH (abbreviated as 'Rossmann' in the following) as accurate as possible.

feature	data type	number of values	EE dimension
store	nominal	1115	10
day of week	ordinal	7	6
day	ordinal	31	10
month	ordinal	12	6
year	ordinal	3 (2013-2015)	2
promotion	binary	2	1
state	nominal	12	6

TABLE I. Features we used from the Kaggle Rossmann competition dataset. *promotion* signals whether or not the store was issuing a promotion on the observation date. *state* corresponds to the German state where the store resides. The last column describes the dimension we used for each entity embedding (EE).

The dataset published by the Rossmann hosts[1] has two parts: the first part is `train.csv` which comprises about 2.5 years of daily sales data for 1115 different Rossmann stores, resulting in a total number of 1017210 records; the second part is `store.csv` which describes some further details about each of these 1115 stores.

Besides the data published by the host, external data was also allowed as long as it was shared on the competition forum. Many features had been proposed by participants of this competition. For example the Kaggle user dune_dweller smartly figured out the German state each store belongs to by correlating the store open variable with the state holiday and school holiday calendar of the German states (state and school holidays differ in Germany from state to state)[2]. Other popular external data was weather data, Google Trends data and even sport events dates.

In our winning solution we used most of the above data, but in this paper the aim is to compare different machine learning methods and not to obtain the very best result. Therefore, to simplify, we use only a small subset of the features (see Table I) and we do not apply any feature engineering.

The dataset is divided into a 90% portion for training, and a 10% portion for testing. We consider both a split leaving the temporal structure of the data intact (i.e., using the first 90% days for training), as well as a random shuffling of the dataset before the training-test split was applied. For shuffled data, the test data shares the same statistical distribution as the training data. More specifically, as the Rossmann dataset has relatively few features compared to the number of samples, the distribution of the test data in the feature space is well represented by the distribution of the training data. The shuffled data is useful for us to benchmark model performance with respect to the pure statistical prediction accuracy. For the time based splitting (i.e. unshuffled data), the test data

[1] `https://www.kaggle.com/c/rossmann-store-sales/data`
[2] `https://www.kaggle.com/c/rossmann-store-sales/forums/t/17048/putting-stores-on-the-map`

is of a future time compared to the training data and the statistical distribution of the test data with respect to time is not exactly sampled by the training data. Therefore, it can measure the model's generalization ability based on what it has learned from the training data.

The code used for this experiment can be found in this github repository[3].

A. Neural networks

In this experiment we use both one-hot encoding and entity embedding to represent input features of neural networks. We use two fully connected layers (1000 and 500 neurons respectively) on top of either the embedding layer or directly on top of the one-hot encoding layer. The fully connected layer uses ReLU activation function. The output layer contains one neuron with sigmoid activation function. No dropout is used as we found that it did not improve the result. We also experimented with a neural network where the entity embedding layer was replaced with an extra fully connected layer (on top of the one-hot encoding layer) of the same size as the sum of all entity embedding components but the result is worse than without this layer. We use the deep learning framework Keras[4] to implement the neural network.

As $Sales$ in the data set spans 4 orders of magnitude, we used $\log(Sale)$ and rescaled it to the same range as the neural network output with $\log(Sale)/\log(Sale_{max})$. Adam optimization method[36] is used to optimize the networks. Each network is trained for 10 epochs. For prediction we use the average result of 5 neural networks, as an individual neural network showed notable variance.

B. Comparison of different methods

We compared k-nearest neighbors (KNN), random forests and gradient boosted trees with neural networks. KNN and random forests are tested using the scikit-learn library of python [37], while we use the xgboost implementation of gradient boosted trees [13]. The used model parameters can be found in Table II. They were empirically found by optimizing the results of the validation set. For the input variables, KNN is fed with one-hot-encoded features, while random forests and gradient boosted trees use the integer coded categorical variables directly. We use $\log(Sales)$ as the target value for all machine learning methods.

As we are using relatively small number of features (7) compared to available training samples (about 1 million) the dataset is not sparse enough for our purpose. Therefore, we sparsified the training data by randomly sampling 200,000 samples out of the training set for benchmarking the models.

xgboost	
max_depth	10
eta	0.02
objective	reg:linear
colsample_bytree	0.7
subsample	0.7
num_round	3000
random forest	
n_estimators	200
max_depth	35
min_samples_split	2
min_samples_leaf	1
KNN	
n_neighbors	10
weights	distance
p	1

TABLE II. Parameters of models used to compare with neural networks. If a parameter is not specified, the default choice of scikit-learn (for random forests and KNN) and xgboost was taken.

Instead of root mean square percentage error (RMSPE) used in the competition we use mean absolute percentage error (MAPE) as the criterion:

$$MAPE = \left\langle \left| \frac{Sales - Sales_{predict}}{Sales} \right| \right\rangle \tag{24}$$

The reason is that we find MAPE is more stable with outliners, which may be caused by factors not included as features in the Rossmann dataset.

The results that we obtained can be found in Table III and IV. We can see that neural networks give the best results for non-shuffled data. For shuffled data, gradient boosted trees with entity embedding (see below for an explanation) and neural networks give comparable good results. Neural networks with one-hot encoding give slightly better results than entity embedding for the shuffled data while entity embedding is clearly better than one-hot encoding for the non-shuffled data. The explanation is that entity embedding, by restricting the network in a much smaller parameter space in a meaningful way, reduces the chance that the network converges to local minimums far from the global minimum. More intuitively, entity embeddings force the network to learn the intrinsic properties of each of the feature as well as the sales distribution in the feature space. One-hot encoding, on the other hand, only learns about the sales distribution. A better understanding of the intrinsic properties of the components (features) will give the model an advantage when facing a new combination of the components not seen during training. We expect this effect will be stronger when we add more features, for both shuffled and unshuffled data.

We also used the entity embeddings learned from a neural network as the input for other machine learning

[3] https://github.com/entron/entity-embedding-rossmann
[4] https://github.com/fchollet/keras

method	MAPE	MAPE (with EE)
KNN	0.315	0.099
random forest	0.167	0.089
gradient boosted trees	0.122	0.071
neural network	0.070	0.070

TABLE III. Comparison of different methods on the Kaggle Rossmann dataset with 10% shuffled data used for testing and 200,000 random samples from the remaining 90% for training.

method	MAPE	MAPE (with EE)
KNN	0.290	0.116
random forest	0.158	0.108
gradient boosted trees	0.152	0.115
neural network	0.101	0.093

TABLE IV. Same as Table IV except the data is not shuffled and the test data is the latest 10% of the data. This result shows the models generalization ability based on what they have learned from the training data.

methods, that is, we feed the embedded features into other machine learning methods. This significantly improves all the methods tested here as shown in the right columns of the tables.

C. Distribution in the embedding space

The main goal of entity embedding is to map similar categories close to each other in the embedding space. A natural question is thus how the embedding space and the distribution of the data within it look like. For the following analyses, we used a store embedding matrix of dimension 50 and trained the network on the full first 90% of data, i.e. we did not apply data sparsification.

To visualize the high dimensional embeddings we used t-SNE[38] to map the embeddings to a 2D space. Fig 3 shows the result for the German state embeddings. Though the algorithm does not know anything about German geography and society, the relative positions of the learned embedding of German states resemble that on the German map surprisingly well! The reason is that the embedding maps states with similar distribution of features, i.e. similar economical and cultural environments, close to each other, while at the same time two geographically neighboring states are likely sharing similar economy and culture. Especially, the three states on the right cluster, namely *Sachsen*, *Thueringen* and *Sachsen Anhalt* are all from eastern Germany while states in the left cluster are from western Germany. This shows the effectiveness of entity embedding for abductive reasoning. It also shows that entity embedding can be used to cluster categorical data. This is a consequence of entity embedding putting similar values close to each other in an euclidean space equipped with distance measure, on which any known clustering algorithm can be applied.

Regarding the sales distribution in entity embeddings,

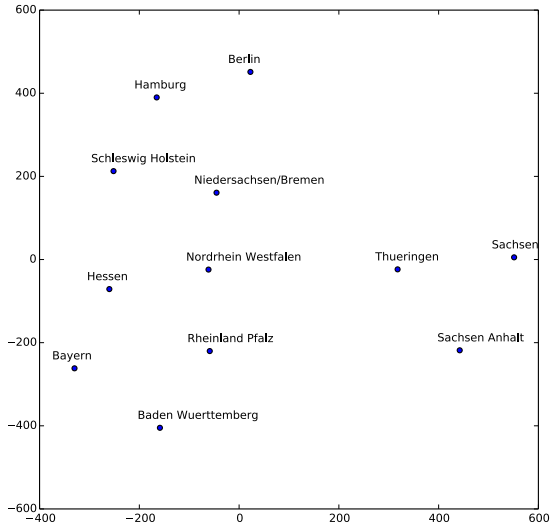

FIG. 3. The learned German state embedding is mapped to a 2D space with t-SNE. The relative positions of German states here resemble that on the real German map surprisingly well.

we take entity embedding of the store as an example. Figure 4 shows the sales distribution in the store embedding along its first two principal components and along two random directions. It is apparent from the plot that the sales follows a continuous functional relationship along the first principal component. This allows the neural network to understand the impact of the store index, as stores with similar sales are mapped close to each other. Although the other directions in the subspace have no direct correlation with sales, they are encoding probably other properties of the store and when combined with other features in the deeper layers of the network they could have an impact on the final sales prediction.

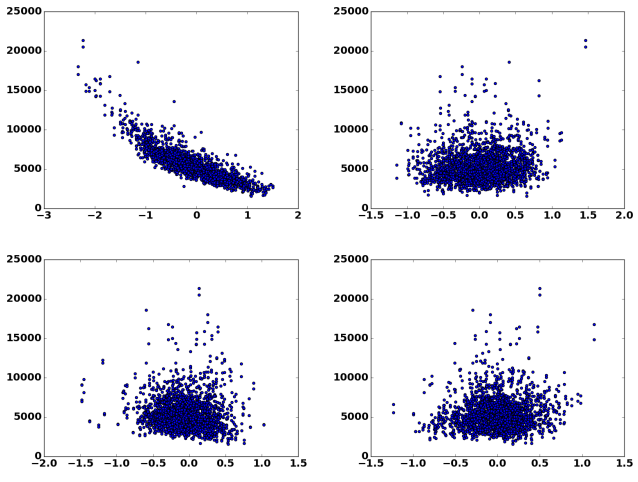

FIG. 4. Sales distribution along first principal component (upper left) and second principal component (upper right) of embedded store indices and along two random directions (lower left and right). All 1115 stores contributed to the plot.

The density distribution of store embedding is visualized in Fig. 5, which shows the distribution along the first four principal components. Interestingly, the univariate density along the first principal components is approximately gaussian distributed. However, their joint distribution is not multivariate gaussian, as the Mardia test [39] reveals.

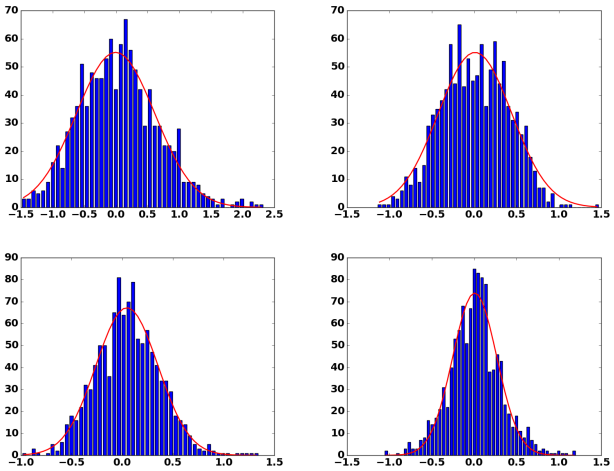

FIG. 5. Density distribution of embedded store indices along the first four principal components (from upper left to lower right). The red line corresponds to a gaussian fit. The p-values of the D'Agostino's K^2 normality test are all statistically significant, i.e. below 0.05.

As can be seen in Fig 1, the neural network is fed with the direct product of all the entity embedding subspaces. We also investigated the statistical properties of this concatenated space. We found that there is no strong correlation between the individual subspaces. It is thus sufficient to consider them independently, as we did in this section.

VII. FUTURE WORK

Due to the limitation of time we leave the following points for future explorations:

First of all, entity embedding should be tested with more datasets, in particular datasets with many high cardinality features, where the data is getting sparse and entity embedding is supposed to show its full strength compared with other methods. For some datasets and entity embeddings it could also be interesting to explore the meaning of the directions in the embeddings like those in Eq. (2) and Eq. (3).

Second, we only touched the surface of the relation of entity embedding with the finite metric spaces. A deeper understanding of this relation might also help to find the optimal dimension of the embedding space and how neural networks work in general.

Third, similar methods may be applied to improve the approximation of continuous (i.e. non-categorical), but non-monotone functions. One way to achieve this is by discretizing the continuous variables and transform them into categorical variables as discussed in this paper.

Last, it might be interesting to systematically compare different activation functions of the entity embedding layer.

VIII. ACKNOWLEDGE

We thank Dirk Rossmann GmbH to allow us to use their data for the publication. We thank Kaggle Inc. for hosting such an interesting competition. We thank Gert Jacobusse for helpful discussions regarding xgboost. We thank Neokami Inc. co-founders Ozel Christo and Andrei Ciobotar for their support joining the competition and writing this paper.

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," Nature **521**, 436–444 (2015).

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems* (2012) pp. 1097–1105.

[3] Matthew D. Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision?ECCV 2014* (Springer, 2014) pp. 818–833.

[4] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556 (2014).

[5] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," arXiv preprint arXiv:1312.6229 (2013).

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015) pp. 1–9.

[7] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara 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**, 82–97 (2012).

[8] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran, "Deep convolutional neural networks for lvcsr," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (IEEE, 2013) pp. 8614–8618.

[9] Yoshua Bengio, Rjean Ducharme, Pascal Vincent, and Christian Janvin, "A neural probabilistic language model," The Journal of Machine Learning Research **3**, 1137–1155 (2003).

[10] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ, "Empirical evaluation and combination of advanced language modeling techniques." in *INTERSPEECH*, s 1 (2011) pp. 605–608.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," .

[12] Yoon Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882 (2014).

[13] Tianqi Chen and Carlos Guestrin, "Xgboost: A scalable tree boosting system," (2016), arXiv:1603.02754.

[14] George Cybenko, "Approximation by superpositions of a sigmoidal function," **2**, 303–314.

[15] Michael Nielsen, "Neural networks and deep learning," (Determination Press, 2015) Chap. 4.

[16] Bernardo Llanas, Sagrario Lantarón, and Francisco J Sáinz, "Constructive approximation of discontinuous functions by neural networks," Neural Processing Letters **27**, 209–226 (2008).

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE **86**, 2278–2324 (1998).

[18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)* (2014) pp. 1532–1543.

[19] Geoffrey E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the eighth annual conference of the cognitive science society*, Vol. 1 (Amherst, MA) p. 12.

[20] Yoshua Bengio and Samy Bengio, "Modeling high-dimensional discrete data with multi-layer neural networks." in *NIPS*, Vol. 99 (1999) pp. 400–406.

[21] Alberto Paccanaro Geoffrey E Hinton, "Learning hierarchical structures with linear relational embedding," in *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference*, Vol. 2 (MIT Press, 2002) p. 857.

[22] Rodolphe Jenatton, Nicolas L Roux, Antoine Bordes, and Guillaume R Obozinski, "A latent factor model for highly multi-relational data," in *Advances in Neural Information Processing Systems* (2012) pp. 3167–3175.

[23] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng, "Embedding entities and relations for learning and inference in knowledge bases," arXiv preprint arXiv:1412.6575 (2014).

[24] Fei Wu, Jun Song, Yi Yang, Xi Li, Zhongfei Zhang, and Yueting Zhuang, "Structured embedding via pairwise relations and long-range interactions in knowledge base," (2015).

[25] Alberto Paccanaro and Geoffrey E. Hinton, "Extracting distributed representations of concepts and relations from positive and negative propositions," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, Vol. 2 (IEEE) pp. 259–264.

[26] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio, "Learning structured embeddings of knowledge bases," in *Conference on Artificial Intelligence*, EPFL-CONF-192344 (2011).

[27] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio, "A semantic matching energy function for learning with multi-relational data," Machine Learning **94**, 233–259 (2014).

[28] Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao, "Learning to represent knowledge graphs with gaussian embedding," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (ACM, 2015) pp. 623–632.

[29] Omer Levy and Yoav Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in Neural Information Processing Systems* (2014) pp. 2177–2185.

[30] Yoon Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882 (2014).

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems* (2013) pp. 3111–3119.

[32] Schoenberg, "Metric spaces and positive definite functions," American Mathematical Society (1938).

[33] Ofer Neiman Ittai Abraham, Yair Bartal, "On embedding of finite metric spaces into hilbert space," Leibniz Center for Research in Computer Science (2006).

[34] Ji? Matouek, "On the distortion required for embedding finite metric spaces into normed spaces," Isreal Journal of Mathematics , 333–344 (1996).

[35] Joseph B Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," Psychometrika **29**, 1–27 (1964).

[36] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," CoRR **abs/1412.6980** (2014).

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research **12**, 2825–2830 (2011).

[38] Laurens Van der Maaten and Geoffrey Hinton, "Visualizing data using t-sne," Journal of Machine Learning Research **9**, 85 (2008).

[39] K.V. Mardia, "Measures of multivariate skewness and kurtosis with applications," Biometrika **57**, 519–530 (1970).