NeuralCubes: Deep Representations for Visual Data Exploration

Zhe Wang
Dept. of Computer Science
University of Arizona
Tucson, AZ. USA
zhew@email.arizona.edu

Dylan Cashman
Dept. of Computer Science
Tufts University
Medford, MA. USA
dylan.cashman@tufts.edu

Mingwei Li
Dept. of Computer Science
University of Arizona
Tucson, AZ. USA
mwli@email.arizona.edu

Jixian Li
Dept. of Computer Science
University of Arizona
Tucson, AZ. USA
jixianli@email.arizona.edu

Matthew Berger
Dept. of Computer Science
Vanderbilt University
Nashville, TN. USA
matthew.berger@vanderbilt.edu

Joshua A. Levine
Dept. of Computer Science
University of Arizona
Tucson, AZ. USA
josh@email.arizona.edu

Remco Chang
Dept. of Computer Science
Tufts University
Medford, MA. USA
remco@cs.tufts.edu

Carlos Scheidegger
Dept. of Computer Science
University of Arizona
Tucson, AZ. USA
cscheid@email.arizona.edu

***Abstract*—Visual exploration of large multi-dimensional datasets has seen tremendous progress in recent years, allowing users to express rich data queries that produce informative visual summaries, all in real time. Techniques based on data cubes are some of the most promising approaches. However, these techniques usually require a large memory footprint for large datasets. To tackle this problem, we present NeuralCubes: neural networks that predict results for aggregate queries, similar to data cubes. NeuralCubes learns a function that takes as input a given query, for instance, a geographic region and temporal interval, and outputs the result of the query. The learned function serves as a real-time, low-memory approximator for aggregation queries. Our models are small enough to be sent to the client side (e.g. the web browser for a web-based application) for evaluation, enabling data exploration of large datasets without database/network connection. We demonstrate the effectiveness of NeuralCubes through extensive experiments on a variety of datasets and discuss how NeuralCubes opens up opportunities for new types of visualization and interaction.**

I. Introduction

Interactive visual exploration is becoming increasingly essential for making sense of large multi-dimensional datasets. It is common for datasets to have billions of data items that contain a variety of attributes of geographic, temporal, and categorical nature. Due to this size and complexity, issuing data queries in real-time is often not feasible as it will result in an unreasonable amount of latency. Instead, efficient data structures are often designed for real-time exploration, in lieu of making raw database queries. These data structures are pre-computed and optimized around queries frequently used by the visualization, such as performing data summaries [33], [45], ranking [38], and applying multivariate statistics [57].

However, while these data structures are effective, they can still be prohibitively large as data size increases. Worse, when data complexity increases (i.e. in terms of the number of dimensions in the data), the sizes of many of these data structures grow exponentially. As a result, these data structures are often stored on a server, and only sub-parts of the data structures are fetched in real-time based on the user's exploration.

Fig. 1. NeuralCubes frames approximate query processing over a database as a neural network, one that takes a query as input, e.g. selection of a temporal interval, and maps it to an aggregation result as output, e.g. a count of the number of records, enabling real-time visual exploration for large-scale data. In the figures above, the blue lines represent visualizations generated with data from the database and the orange lines are generated using NeuralCubes. Note that the two are nearly indistinguishable, but NeuralCubes uses only a fraction of the memory footprint as the database.

In this paper, we introduce NeuralCubes, a technique that approximates queries on multi-dimensional datasets for the support of interactive visual exploration. NeuralCubes is a deep neural network that is compact in storage, can respond to data queries in real time, and is sufficiently accurate for visualization purposes. Thus, unlike existing systems that have high storage requirements and require intricate data structures [33], [34],

978-1-6654-3902-2/21/$31.00 © 2021 IEEE

NeuralCubes eliminates the need for a visualization system to fetch data or sub-parts of a data structure from a server. Instead, NeuralCubes can be stored in a client's memory and is easily accessible for use in visualization applications. In particular, NeuralCubes supports Selection, Projection, and Aggregation (SPA) queries – operations that are common to visual data exploration systems that rely on query patterns to achieve gains in performance and storage (e.g. imMens [34] and Nanocubes [33]).

NeuralCubes frames the action of querying a database as a learnable function, one that maps from a given input query to an aggregation result. Assuming that there are latent patterns in the data (i.e. that the data is not purely random), these mappings can be effectively learned. In particular, we observe that typical visualizations generate a limited number of query templates and expect a fixed number of numeric values in response. For example, in Fig. 1 the query to the database is based on four sets of filters (geographic region, month of year, day of week, and time of day). In response, the visualization anticipates a set of numeric values to populate the geographic heatmap and the three line charts.

To train NeuralCubes, we first need an application model and a user model to generate a training set. An application model can be seen as the "data schema" of an application. It contains information on the number and the types of data attributes used to generate the visualization. A user model derives from types of queries and the frequency with which they are issued from users. With these two models, we can generate query-result pairs as a training set to optimize NeuralCubes. Once trained, the learned neural network can answer any queries issued from the same application.

We evaluate NeuralCubes on a variety of datasets including BrightKite social network check-ins [7], Flights dataset [42], YellowCab taxi dataset [53], and SPLOM dataset [25]. We quantitatively analyze the accuracy of NeuralCubes and how network size, training set size, raw data size, and attribute resolution affect prediction. Across these datasets, we demonstrate that we can achieve approximately 3% Relative Absolute Error (RAE) with a model less than 1MB for datasets containing millions of records. Since the model is very small, these NeuralCubes models can be evaluated in real-time on thin web browser clients and support interactive visualization and exploration of large amounts of data with low latency.

Lastly, we observe that the trained neural network model can be used to aid a user's understanding of the patterns in the data. Since the model learns the relationship between the input vectors and the output query results, visualizing that relationship can reveal the potential underlying structures in the data. In section IV we present visualizations of the latent layers of a NeuralCubes model and how the visualizations can help a user better make sense of their data.

To summarize, the contribution of this work includes:

- We show that neural networks can learn to answer aggregate queries efficiently and effectively, that they generalize across heterogeneous attribute types (such as geographic, temporal, and categorical data), and present a method to convert the schemata needed to describe visual exploration systems into an appropriate deep neural network architecture;
- We use these neural networks that learn the structure of aggregation queries to provide the user 2D projections that enable the intuitive exploration of data queries; and
- We conduct extensive experiments on a variety of datasets that proves the effectiveness of our approach.

II. Related Work

Our work proceeds from recent work in two mostly disparate fields; data management and neural networks. In data management, we discuss architectures, data structures, and algorithms that exploit access patterns to offer better performance. In neural networks, we review some of the recent applications of neural networks to novel domains, as well as relevant work on the interpretability of deep networks.

A. Data management

The importance of data management technology in the context of interactive data exploration has been recognized for over 30 years, with the work of MacKinlay, Stolte, and collaborators in Polaris [35], APT [50], and Show Me [36] being central contributions to the field. Since then, researchers in both data management and visualization have extended the capabilities of data exploration systems (both visual and otherwise) in a number of ways. General-purpose database systems now exist for fast query processing of large amounts of data with approximate answers. Commonly referred to as Approximate Query Processing (AQP), AQP systems use a variety of techniques to achieve low latency, including offline sampling (e.g. [1], [10], [32], [46]), online aggregation (e.g. [9], [20], [23], [62]), and probabilistic models (e.g. [5], [43], [59]). Specific to visualization, Wu et al. proposed techniques to tailor databases for large-scale data visualization [2], [58]. NeuralCubes, as we will later discuss, provides evidence that machine learning techniques should *also* be designed with visualization in mind, and that such design enables novel visual data exploration tools.

While we developed NeuralCubes to leverage machine learning (ML) for providing richer information during data exploration itself, we are not the first to propose the use machine learning in the context of data management. Notably, ML has been recently used to enable *predictive interaction*: if a system can accurately predict the future behavior of the user, there are ample opportunities for performance gains (and specifically for hiding latency) [3], [6].

Gray et al.'s breakthrough idea of organizing aggregation queries in the appropriate lattice—the now-ubiquitous *data cube*—spawned an entire subfield of advances in algorithms and data structures [18], [19], [49]. This work has gained renewed interest in the context of interactive exploration, where additional information (such as screen resolution, visualization encoding, and query prediction) can be leveraged [24], [33], [34], [45]. Since every query in NeuralCubes is executed by a *fixed-size network*, it also provides low latency in aggregation queries. But because NeuralCubes

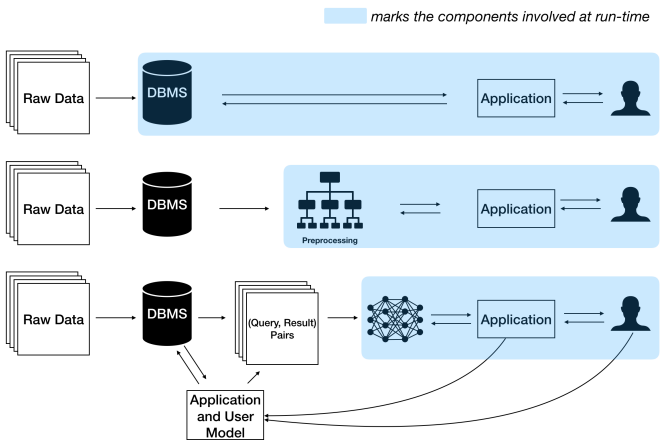

Fig. 2. Comparison of systems using traditional database, Datacubes and NeuralCubes.

models learn the interaction between query inputs and results, the model can provide additional information about the dataset that can further aid interactive exploration.

B. Deep Neural Networks

Our approach is inspired by the recent success of applying deep neural networks to a variety of domains, including image recognition [31], machine translation [51], and speech recognition [21]. These techniques are solely focused on prediction, and our method is similar, in that we are focused on training deep networks for the purposes of query prediction. Yet, we differ in that prediction is not the only goal, rather we want to perform learning in a manner that provides the user a fast and low-memory-cost way to visually explore data. The query prediction task at hand can be viewed as a means to realize these goals.

In this context, our method for learning 2D latent space of queries can be viewed as a form of unsupervised learning, where treating query prediction as pretext, the features that we learn along the way can be used for other purposes – in our case exploratory purposes. This is similar to recent techniques in computer vision that learn features using different forms of self supervision, for instance learning to predict spatial context [11], [40], temporal context [56], and perhaps more pertinent to our work, learning to count visual primitives in a scene [41]. These techniques solve certain types of relevant visual tasks that do not require human supervision, but then extract the learned features for supervised learning. Our approach is similar: our training data does not require human intervention, since it is built from existing data cubes techniques, yet the features that we learn from this task can be used to help with visual data exploration.

Last, we note that there is some recent work that seeks to combine databases with neural networks. Thirumuruganathan et al. [54] use generative neural networks to perform data sampling for approximate query processing. Kipf et al. [28] and Ortiz et al. [44] use trained neural networks to estimate query cardinalities. Kraska et al. [30] make the connection between indexing, such as b-trees or hashes, and models, and show that such indexing schemes can be learned using neural networks. Mitzenmacher [39] consider similar learning techniques for Bloom filters. These methods are concerned with using neural networks to speed up computation and minimize memory storage. Although we demonstrate that our method can attain these benefits, the primary focus of our method is in using a neural network as an integral component to visual exploration, i.e. NeuralCubes is not trying to predict *all* queries that a database can answer, but rather only the subset of queries that are applicable to a given visualization.

III. NeuralCubes: Replacing a Database with a Learned Neural Network

We introduce our approach by comparing it with visualization systems that utilize traditional databases and those using advanced preprocessing techniques. Fig. 2 shows an overview of these different approaches.

First, we briefly discuss the data queries that are required by interactive visualization systems. Suppose we are given a set of *records*, each record contains a set of *attributes*, and each attribute has a certain *type* (e.g. continuous, categorical, geographic, temporal, etc.) that characterizes the set of *values* it may take on. Database queries may return a single record, or multiple records, and in the case of the latter it is often of interest to summarize the set of records by performing an *aggregation*, for instance count, average, or max, depending on the attribute type. Within a visualization system, the set of attributes, their types, and the class of aggregations determine the sorts of queries one may issue that serve as the backbone for visual interaction. For instance, if the attribute type is categorical or temporal, then we can visualize this result as a histogram, whereas if the attribute type is geographic, we may plot the result as a heatmap over a spatial region.

Naïvely, a visualization system can directly issue SQL queries to the database to get the needed information. Shown as the top row in Fig. 3, the advantages to this approach are that it supports "cold start" – where no additional data structure and or pre-processing is required before the application can retrieve data from the database. However, this approach does not scale well when the dataset gets larger, resulting in long latency in query processing which limits its practical use in interactive visualization systems for large datasets.

To reduce latency, a common technique is to pre-compute an auxiliary data structure such as a data cube (shown as the second row in Fig. 3). During run-time, the visualization application only interacts with this data structure without directly querying the database. While this approach has been effective in reducing query latency, the size of the data cube can grow exponentially as the number of data attributes increases. Not only does this result in a large memory footprint, the preprocessing time can also become prohibitively high as the dataset grows in size and attributes.

The basic idea behind NeuralCubes is the use of neural networks to learn the process of performing database aggregation queries. We treat a neural network as a function that

	"Cold Start"	Small Memory Footprint	Low run-time latency	Controllable error
Regular Database	+	+	−	+
Database with preprocessing	−	−	+	~
NeuralCubes	−	+	+	~

"+" means support the feature, "−" means don't support the feature, "~" means it depends on specific techniques

Fig. 3. Comparison of supported features of different approaches.

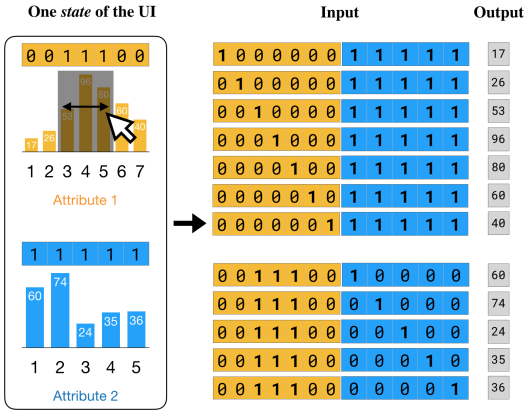

Fig. 4. The corresponding input-output pairs for one *state* of the UI. For a given attribute, a query is encoded by forming a vector over attribute values, assigning 1 to a value that is selected in the query, and 0 otherwise.

approximates a database aggregation query, where the input is a data query in the form of a set of attribute ranges, and the output is the aggregation of the data returned from the given input query. Unlike existing techniques [33], [45], [57], it is thus necessary for NeuralCubes to learn by example, rather than precomputing data structures for query processing. Specifically, in order to train our model we need to gather query inputs, and their corresponding aggregation outputs. Although this can lead to an exponentially-large number of possible examples, we need only be concerned with queries that are reflective of user behavior, e.g. actions that a user would perform within a visualization. This visualization-first principle enables us to drastically limit the number of examples necessary to train our model for good generalization. Moreover, this enables extremely compressive model sizes, relative to the datasets, and as a consequence real-time querying that supports interactive exploration.

On the other hand, the use of neural networks implies that we cannot ensure strict error bounds. Yet in the context of AQP, we believe neural networks are ideal for visual exploration, as absolute errors are negligible so long as trends and patterns in the visualization are faithfully preserved. We summarize the trade-offs of different approaches discussed above in Fig. 3.

A. What's the input of NeuralCubes?

We first define the concept of *state* of a data visualization system. We assume that the underlying database schema has a total of d attributes, where we denote each attribute by $a_i, 1 \leq i \leq d$, and we represent the *range selection* operation for a given attribute a_d by $r(a_d)$. For instance, if an attribute was hour-of-day, then the range operation on this attribute would return a set of hours. At any time, there must be a range selected on each attribute – we treat the absence of an attribute's selection as all of its values being selected. We call this set of ranges a *state* of the visualization system, denoted as $S = r(a_1), r(a_2), ...r(a_d)$. The corresponding query results are $DB(S) \in \mathbb{R}$. Our objective is to train a neural network $f(S) \in \mathbb{R}$ to best approximate DB, given *training data* $D = ((S_1, DB(S_1)), (S_2, DB(S_2)), ...(S_n, DB(S_n)))$, where S_i is a state, i.e. $S_i = (r_i(a_1), r_i(a_2), ...r_i(a_d))$, and $DB(S_n)$ is the aggregation result from the database. For example, the state shown in Fig. 4 is $S = \{[3,5],[1,5]\}$.

What remains is a method for representing the set of range selections as input for the neural network. This is nontrivial due to the different types of attributes, e.g. geographic, temporal, categorical, as well as the types of selections that can be performed on attributes, e.g. spatially contiguous selections in geographic coordinates. To address these challenges in a unified manner, we use *many-hot encodings* for attribute selections, as shown in Fig. 4. Many-hot encodings are generalizations of one-hot encodings, commonly used as a way to uniquely represent words in neural language models [4], categorical inputs for generative models [12], as well as geographic coordinates for image recognition [52].

More specifically, for a given attribute a_i we assume that it may be discretized into $m(a_i)$ many values. For certain attributes, this assumption is natural: categorical data, temporal data such as hour-of-day or day-of-week, while for continuously-valued data we uniformly discretize the data space into bins. For the attribute's selection $r(a_i)$, we then associate a binary vector $\mathbf{r}(a_i) \in \{0,1\}^{m(a_i)}$ such that $\mathbf{r}(a_i)_j = 1$ if the value at index j belongs to the selection, and 0 otherwise. This permits arbitrary types of selections for categorical and temporal data. Spatial data, specifically 2D geographic regions, is slightly more complicated: one option is to represent each discretized cell as a single dimension in $\mathbf{r}$, but this would result in a large number of inputs for even small spatial resolutions. We simplify this problem by restricting selections on 2D regions to be rectangular. Thus to represent such a region we associate a pair of vectors $\mathbf{r}^x(a_i)$ and $\mathbf{r}^y(a_i)$ for the selected x and y intervals, respectively, of the rectangle and then concatenate these two vectors to form the input. In practice, non-rectangular selections can be approximated by issuing multiple rectangular selections.

B. Generating Training Data: Modeling Application and User Interaction

The data used to train our model is intended to reflect common user actions within visual exploration. This is necessary to ensure that our network will properly generalize. To this end,

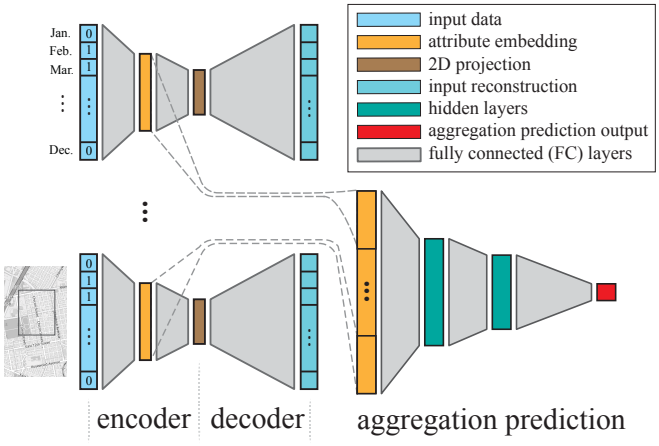

Fig. 5. We highlight the general structure of our neural network. For each attribute we learn a feature embedding (orange), and then use the embeddings for two purposes: we fuse the embeddings through a feature-wise concatenation, which is fed into an MLP for aggregation prediction (red), as well as learn attribute-specific autoencoders to derive a 2D latent space (brown).

we simulate user actions on a specific interface and use the corresponding queries for training. Specifically, as discussed in section III-A, we should randomly generate a *state* of the UI and then turn it into actual queries. Thus, to generate training samples, we first generate a range selection for each attribute, e.g. contiguous ranges for temporal or spatial attributes. Then we perform a group by query on one of the attributes with the constrains (range selections) on other attributes, resulting in a batch of query-result pairs for the current attribute. We do the same for every attribute, thus giving us all query-result pairs of a *state*.

Given this setting of query generation, we propose the following strategy for simulating user queries. For a specified attribute, we uniformly sample the length of its range from all valid lengths and then uniformly choose a start and end position. Using *month* as an example, the possible length of ranges we can take are in the interval $[1,12]$. Suppose we randomly choose a length, say 3. Given this length, the possible inclusive starting points/lower bounds are $[1,10]$. A random selection from this set, say 2, will result in a range selection of $[2,5]$.

While it may seem artificial to carefully sample a training set to make the network fit a certain kind of input, it is important to remember that NeuralCubes is designed, first and foremost, with visualization in mind. Thus, even if we are not necessarily learning over the full data distribution of queries, so long as our sampling resembles the manner in which users perform selection, then a user's interaction with the network should remain meaningful.

C. A Neural Network Architecture for Data Queries

NeuralCubes is designed as a multi-layer perceptron, where separate representations are learned for each attribute, and finally combined to predict an aggregation query. The intuition behind this architecture is to ensure attribute representations are *predictive of aggregation queries*, providing us a more informative representation than the input many-hot encodings, and then to combine these features to learn their relationships in predicting the aggregation. We use this general architecture for all of the datasets in the paper, shown in Fig. 5, but tailor the architectures based on the given dataset, which we defer to Section 6. All networks, nonetheless, share the following steps to form the network f:

1) **Learning Attribute Embeddings.** For a given set of attribute selections represented as binary vectors, we first transform each of them separately into their own feature embedding. Namely, for attribute a_i, let $f_i : \mathbb{R}^{m(a_i)} \rightarrow \mathbb{R}^{d_i}$ represent a series of layers that transforms the attribute selection to a d_i-dimensional embedding space.
2) **Attribute Embedding Concatenation.** We then concatenate the embeddings into a single vector $\hat{\mathbf{f}} = [f_1(\mathbf{r}(a_1)), f_2(\mathbf{r}(a_2)), ...f_d(\mathbf{r}(a_d))]$, where $\hat{\mathbf{f}} \in \mathbb{R}^{\hat{d}}$, $\hat{d} = \sum_{i=1}^{d} d_i$.
3) **Aggregation Query Prediction.** Given the concatenated embedding $\hat{\mathbf{f}}$, we then feed it through a series of layers, where the last layer outputs a single value, corresponding to the aggregation query. Multiple fully connected layers are used in order to learn the relationship between the attributes, so as to make better predictions.

Prediction Loss. Given the neural network f, we can now optimize over its set of parameters to best predict database queries *DB*. For this purpose, we define a loss function for prediction that combines an L1 loss and a mean-squared loss for a given query Q:

$$L_{pred} = \lambda_1 |f(Q) - DB(Q)| + \lambda_2 (f(Q) - DB(Q))^2, \quad (1)$$

where λ_1 and λ_2 weight the contributions of the L1 and mean-squared losses, respectively. The intuition behind this loss is to learn the general trend in the data, captured by the L2 loss, but in order for the training to not be overwhelmed by aggregations that result in very large values, the L1 loss provides a form of robustness.

1) Autoencoder: Reconstruction as Regularization: In order to ensure NeuralCubes does not overfit to the training data, we introduce a reconstruction-based regularization for individual attributes. Reconstruction as regularization has been used in many recent works [27], [47], [60]. We think this approach aligns well with our main goal: we would like the model to learn the underlying data distribution than memorize the correlation between the noise in the input and the corresponding output. Also, the reconstruction step can provide opportunities for new types of visualization, which we will discuss more in section IV-C.

We achieve this by defining an *autoencoder* [22] for each attribute query a_i. More specifically, we learn a projection to a 2D latent space via an MLP, starting from the input layer, going to 2D, denoted as an *encoder* by $\mathbf{e}_i : \mathbb{R}^{d_i} \rightarrow \mathbb{R}^2$. We also want to project back: reconstruct the original query (via its binary representation) from its 2D position, or a *decoder* $\mathbf{d}_i : \mathbb{R}^2 \rightarrow \mathbb{R}^{m(a_i)}$. Regularization is achieved by weight sharing: the first few layers in the encoder will be shared with the

regressor, shown in Fig. 5. We want to emphasize that the 2D harsh-bottleneck layers are not used in making predictions. They exist to autoencode the input query and provide us a visual representation of the data. Only earlier wide layers are connected with the regressor. The learned attribute embedding is a sufficiently large subspace to regularize and improve the predictions.

Autoencoder Loss. Since we represent attribute selections as binary-valued, a suitable loss function for measuring the quality of our autoencoder is the binary cross entropy loss:

$$L_{ae} = -\sum_{j=1}^{m(a_i)} (\mathbf{r}(a_i)_j \log(\mathbf{d}_i(\mathbf{e_i}(z_i))_j) + (1-\mathbf{r}(a_i)_j)\log(1-\mathbf{d}_i(\mathbf{e_i}(z_i))_j)), \quad (2)$$

where $z_i = f_i(\mathbf{r}(a_i))$ is the feature embedding of the query selection. Note that this loss is defined for each attribute, in order to learn attribute-specific autoencoders.

2) Combining Prediction Loss and AE Loss Together: We combine the prediction loss and the autoencoder loss to learn a function that can *both* predict queries as well as learn 2D projections of attributes:

$$L = L_{pred} + \lambda_3 L_{ae}, \quad (3)$$

where λ_3 is a weight giving importance to the autoencoder, relative to the weights on the prediction loss. One can view this objective as a type of multi-task autoencoder [17]: we want to learn an embedding, and a 2D projection, that enables self-reconstruction, while simultaneously learning to predict query aggregations. Importantly, this permits us to *contextualize* attribute selections with respect to the aggregation task. The prediction task can be viewed as a form of supervision for the 2D projection task, thus attribute selections that result in similar predictions will have similar feature embeddings, as well as similar 2D projections.

IV. Using NeuralCubes for Visual Exploration

In this section, we describe how NeuralCubes can be used to build interactive data visualization systems.

A. Plotting Histograms and Heatmaps with NeuralCubes

In traditional data cubes techniques, queries are typically made in order to plot histograms (for 1D attributes) and heatmaps (for 2D attributes). This is typically realized through group by queries, where selections are made for all but one attribute, and then for the held-out attribute, a single query is made to gather aggregations for each of its values, i.e.

```
SELECT COUNT(*) FROM BrightkiteTable
GROUP BY dayofweek
```

NeuralCubes can enable the same type of visual exploration. More specifically, we perform a group by query through our many-hot input encoding, placing a 1 on the attribute value that we would like to query, and a 0 for all other attribute values. Furthermore, we can take advantage of GPU data-parallelism in neural network implementations, and perform this operation in a single mini-batch, providing a significant speed-up through GPU acceleration. Our interface allows the user to perform arbitrary range selections for a given attribute, and enables interactive updates of histograms/heatmaps over the remaining attributes, see Fig. 6 for an illustration.

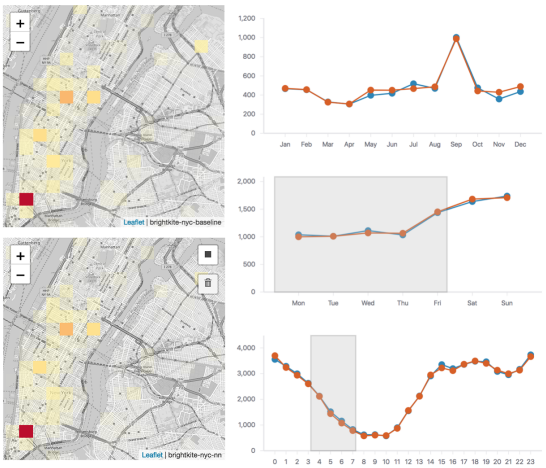

Fig. 6. NeuralCubes can be used in a similar fashion to traditional data cubes techniques, allowing us to plot histograms and heatmaps with respect to various attribute selections.

B. Evaluation at Client Side

Another advantage of NeuralCubes is that the trained model is small enough to be sent to client side for evaluation. In comparison, other OLAP datacubes based techniques require network connections with a backend server for interaction. This advantage of NeuralCubes can be beneficial to both system users and service providers. First, users can expect better experience when making queries. Being able to evaluate at client side, NeuralCubes can eliminate network latency, which is usually a bottleneck. Secondly, service providers can expect much lower cost because the same server can provide service to far more users since Queries Per Second (QPS) will be significantly lower than other client-server systems.

C. Visualizing Attribute Latent Spaces

Although the network can replicate the types of queries perform with data cubes techniques, we can also use different structures that the network learned to enable new forms of visual exploration. In particular, we allow the user to explore the space of attribute selections through each attribute's learned 2D projection, as discussed in Section III-C1. To enable this, we first generate all possible ranges selections for a given attribute, and use the autoencoder to create an overview of their distribution in the 2D space, where we visually encode attributes in a scatterplot. Each point in the latent space view represents a selection of this attribute, where the radius of the point is proportional to its aggregation value, and the color of the point represents the range of the selection, namely the number of values selected in the attribute. Importantly, we ensure that the latent space and the histograms/heatmaps

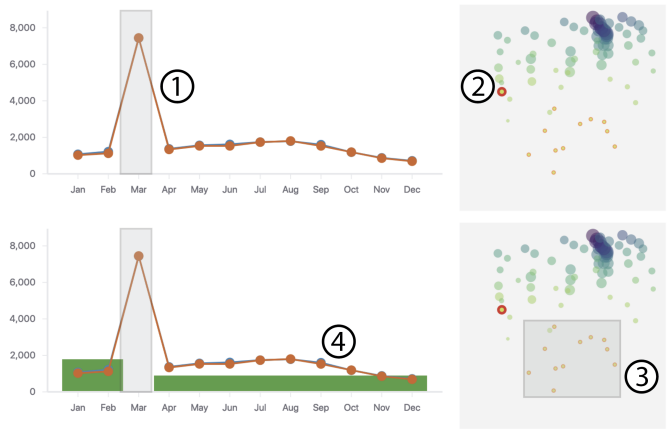

Fig. 7. We show how the user can jointly interact with traditional histogram visualizations and latent space visualizations learned by NeuralCubes. On the left we initially show an histogram of counts of social media check-ins dataset of Austin for attribute *Month*. Upon selecting month *March* (1), the latent space on the right updates, to reflect this selection (2) (colored by red circular stroke with larger width) and also highlight (colored by red circular stroke with smaller width) all possible selections with the same length of range (which is 1 in this case). By performing a selection in the latent space (3), the user can explore the frequency of attribute selections (4) that belong to the selected latent space subset.

discussed previously in Sec. IV-A are *linked*, so that interactions in one view update the other view, see Fig. 7 for an example.

V. Implementation Details

Software NeuralCubes's software is designed to dynamically generate neural network architecture via specification of a JSON configuration file. When changing a dataset or updating the neural networks, users only need to provide a new JSON configuration. For a given trained model, we utilize it in a Flask http backend server, providing RESTful web services. To test the functionality of evaluating at client side, we converted the model from PyTorch format to TensorFlow format. Then we use TensorFlow.js in client side for model evaluation. We have implemented a web user interface, NeuralCubes Viewer, for interaction, implemented in Javascript using React and React-Vis.

Hardware All the models are trained on a machine with an 8 core Intel i7-7700K 4.20GHz CPU, 32GB main memory, and a Nvidia GTX 1080 Ti GPU with 12GB video memory.

Datasets A summary of datasets and training/testing statistics of all the case studies is provided in Table II. In general, each training set contains about 10^4 *states* while the number of all possible states in our experiments is in the order of 10^{10}. We evaluated our method on hold-out test datasets, generated in the same manner as training data. Each testing set contains 1000 *states*. Testing error is computed as the average L1 norm difference between the predictions and ground truth, scaled by the inverse of the mean of the ground truth set, in order to be commensurable across datasets.

Architectures Table I describes the architectures used in each of the trained models we discuss in this paper.

TABLE I
Architectures of the Neural networks Used in the Experiments

Input	Autoencoder	Regressor
BrightKite		
Month (12)	[8, **4**, 2, 4, 8]	[220]
Day of Week (7)	[8, **4**, 2, 4, 8]	
Hour (24)	[12, **6**, 2, 6, 12]	
Geospatial (40)	[400, **128**, 2, 128, 400]	
Flights (count)		
Month (12)	[120, **20**, 2, 20, 120]	[256, 128]
Day of Week (7)	[70, **20**, 2, 20, 70]	
Hour (24)	[240, **20**, 2, 20, 240]	
Geospatial (40)	[400, 128, **20**, 2, 20, 128, 400]	
Carrier (10)	[100, **20**, 2, 20, 100]	
DelayBin (14)	[140, **32**, 2, 32, 140]	
Flights (delay)		
Month (12)	[120, **20**, 2, 20, 120]	[256, 128, 64]
Day of Week (7)	[70, **20**, 2, 20, 70]	
Hour (24)	[240, **20**, 2, 20, 240]	
Geospatial (40)	[400, 128, **20**, 2, 20, 128, 400]	
Carrier (10)	[100, **20**, 2, 20, 100]	
DelayBin (14)	[140, **32**, 2, 32, 140]	
Yellow Cab		
Month (12)	[120, **20**, 2, 20, 120]	[220]
Day of Week (7)	[70, **20**, 2, 20, 70]	
Hour (24)	[240, **20**, 2, 20, 240]	
Geospatial (40)	[400, 128, **20**, 2, 20, 128, 400]	
SPLOM		
a0 (#bin)	[16, **8**, 2, 8, 16]	[120, 60]
a1 (#bin)	[16, **8**, 2, 8, 16]	
a2 (#bin)	[16, **8**, 2, 8, 16]	
a3 (#bin)	[16, **8**, 2, 8, 16]	
a3 (#bin)	[16, **8**, 2, 8, 16]	

Bold numbers represent the attribute embedding layers as described in Fig. 5.

Training Our loss function requires the specification of 3 hyperparameters, namely, the L1 loss (λ_1), MSE loss (λ_2), and AE loss (λ_3). In practice we found L1 loss to be of highest importance, followed by the AE loss, while the MSE loss needed to be quite small just to ensure stable training - please see Sec. VI for dataset-specific hyperparameter settings. For optimization, we use Adam [26] for the first 15 epochs, and switch to standard mini-batch stochastic gradient descent afterwards. Furthermore, we ensure that minibatches are comprised of a balanced distribution of attributes, in order to not bias attributes that offer a large range of possible queries, e.g. geographic queries (2D) vs. temporal queries (1D).

Testing To evaluate the accuracy of NeuralCubes, for each dataset we generate a withheld test set consisting of randomly-generated *states* of the user interface, and their corresponding queries. In our experiments, we use 1000 *states* for testing. Since the ultimate goal of NeuralCubes is to provide an approximation view (e.g. a bar chart), we choose relative absolute error (RAE) as our error metric. RAE is expressed as: $RAE = \frac{\sum_i |\hat{y}_i - y_i|}{\sum_i |y_i - \bar{y}|} \times 100\%$, where $\hat{y}_i$ is the model's prediction for query i and y_i is the ground truth for query i. We choose RAE as it is more aligned with how humans perceive visualizations than MSE. Specifically, humans are more sensitive to relative differences in graphical marks [8], e.g. ratio assessments given the heights of two bars in a bar chart.

VI. Datasets

A. Brightkite Social Media Check-ins

We use the Brightkite [7] social media check-ins dataset to assess the capability of NeuralCubes to learn the count

aggregation. The Brightkite [7] dataset contains social network check-in time and location information. Fig. 8 shows examples of how our NeuralCubes can be used as a data cubes system, plotting histograms and heatmaps from aggregation queries.

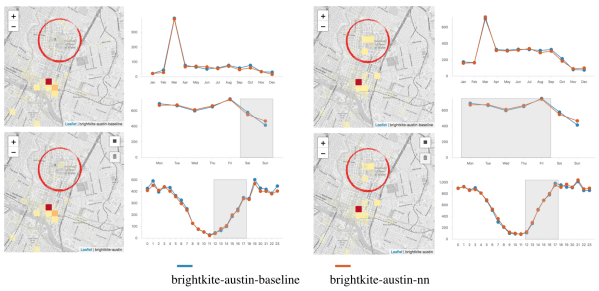

Fig. 8. NeuralCubes learns more check-ins on University of Texas at Austin campus are during daytime on weekdays (right) rather than weekends (left).

We choose two separate metropolitan areas and train separate networks on each: New York City and Austin. We choose month, day of week, hour, and geospatial information (longitude and latitude) as input dimensions, following common practice in the study of urban activity [33], [37]. The longitude and latitude are encoded as $20+20=40$ bins, following the strategy described in Fig. 4. The weights for L1 loss and L2 loss for regressor and binary cross entropy (BCE) loss for autoencoder are 20, 0.001, and 1 respectively. Each model is trained for 1000 epochs and each epoch takes 6 seconds to train.

B. Flight Dataset

We use a dataset collected by the Bureau of Transportation Statistics consisting of flight delay information in 2008 [42] that contains airplane arrival information. For this dataset, our first experiment uses the total flight counts as the aggregation operation. Since this dataset contains an attribute *delay time*, which is also a meaningful attribute in which to aggregate, our second experiment builds a model to predict the average delay time. Our goal with this model is to check the extent to which NeuralCubes can learn non-monotonic aggregations.

Count Predictions For the count aggregation we filter flights to be within the contiguous United States, and restrict entries to only the 10 most used airlines in the dataset, giving us a total of 5,092,321 entries after removing entries containing missing data. This dataset has more entries and attributes than the Brightkite dataset, including a numeric variable (Delay Time). To encode the numeric variable in our many-hot encoding, we bin the delays in 15 minute increments. The weights for L1 loss, L2 loss and BCE loss are 1, 1e-7, and 1, respectively. Each model is trained for 500 epochs and each epoch takes 160 seconds to train.

Average Predictions We follow a similar training setup to count. Since generating training samples to predict average delay time itself is very time consuming, we dropped longitude and latitude columns in the raw data and discarded entries whose delay time is smaller than -60 minutes or larger than 140 minutes. This yields the final dataset with 5,013,088 entries. The weights for L1 loss, L2 loss and BCE loss are 10, 10, and 1, respectively. Each model is trained for 500 epochs and each epoch takes 160 seconds.

TABLE II
SUMMARY OF TRAINING RESULTS

DataSet	Raw Data Size	# States	M. Size	RAE
B.K. NYC	79k (3.1MB)	10k	703KB	4.25%
B.K Austin	22k (0.8MB)	10k	703KB	3.88%
Flights_Count	5m (204MB)	60k	1.2MB	3.11%
Flights_Delay	5m (204MB)	60k	1.2MB	6.58%
Yellow Cab	12m (1.8GB)	30k	798KB	0.97%
SPLOM	100k (3.9MB)	10k	135KB	2.64%

(*# States* means the number of *states* used in the training set. *M. Size* represents the saved file size of trained models.)

C. YellowCab Taxi Dataset

We use NYC YellowCab Taxi trip records of year 2015 from NYC Taxi and Limousine Commission (TLC) [53] to study the learning capacity of NeuralCubes in a series of controlled settings. This dataset contains taxi pickup location and time information in NYC.

We choose month, day of week, hour, and pickup location (longitude and latitude) as input dimensions. The longitude and latitude are encoded as $20+20=40$ bins. We created four different datasets under this same schema by sampling 1k records per month, 10k records per month, 100k records per month and 1 million records per month respectively from the original dataset. We refer to these four datasets as YC-1K, YC-10K, YC-100K, and YC-1M respectively. (To avoid unnecessary data processing, we performed data cleaning and filtering after sampling resulting in a dataset with slightly fewer records per month.) As described in Table I, we use the same network configuration for the four datasets. The weights for L1 loss, L2 loss and autoencoder loss are 100.0, 0.0 and 1.0 for YC-1K, YC-10K and YC-100K. For YC-1M, the weights for L1 loss, L2 loss and autoencoder loss are 10.0, 0.0 and 1.0. Each model is trained for 1000 epochs and each epoch takes 15 seconds.

D. SPLOM Dataset

Last, we use the synthetic SPLOM dataset of Kandel et al. [25] to validate whether NeuralCubes can learn how to predict aggregational values under a controlled setting. This dataset contains five real-valued attributes. The attributes are designed to be correlated. Since all the attributes of SPLOM dataset are real values, it also provides us an opportunity to study the behavior of NeuralCubes when bin size increases. We generated $100,000$ entries of five-attribute records, and divide each attribute into a prescribed number of bins. We trained five different NeuralCubes using 10, 20, 30, 40, and 50 bins respectively. The weights for L1 loss, L2 loss and autoencoder loss are 1.0, 0.0 and 1.0, respectively. Each model is trained for 500 epochs and one epoch takes 7 seconds on average.

VII. Evaluation Results

A. Accuracy, Query Time, and Training Stability

We show the testing RAE for all the datasets in Table II. Most of the prediction RAE are under 5%. The highest is 6.58% for the Flight dataset when predicting *average delay time*, which is more complicated than *count*. This shows the generalizability of NeuralCubes to different data modalities.

We also tested aggregation queries on two different attributes on Flight dataset: *count* and *average delay*. Table II shows the quantitative results for the two types of aggregation queries. Overall, we find the errors to be comparative, showing that our method is capable of handling different types of attributes and different forms of aggregation.

We also measured the query time when using trained models for approximation. To compared with *Group By* queries supported by many data management systems, we do not simply measure the query time for one many-hot input; but instead measure the set of many-hot queries needed to answer a *Group By* query. Specifically, we measure *Group By* queries for *Day of Week*, *Hour*, and *Month*, for Brightkite NYC dataset and Yellow Cab dataset. For the Brightkite NYC data, group by queries take 0.051s, 0.065s, and 0.076s respectively for day, hour, and month. For Yellow Cab, they follow a similar pattern taking 0.014s, 0.020s, and 0.034s respectively.

We evaluate the training stability on BrightKite NYC dataset. Specifically, we independently run 50 training on BrightKite NYC dataset with exactly the same configurations. Then we record the testing error for each model for each epoch. The results are shown in Fig. 9. We can see that NeuralCubes converges to local optima of similar performance.

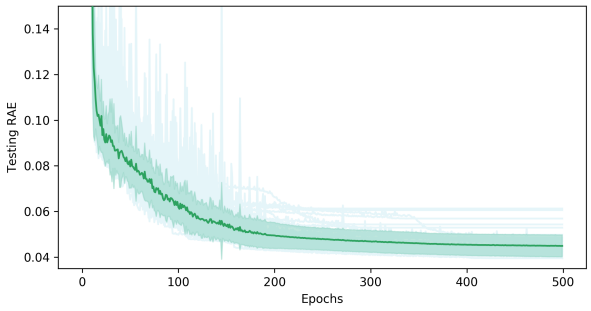

Fig. 9. Testing error for 50 independent training on BrightKite NYC dataset with the same configuration. Light green lines represent each individual training. Solid green line represent the mean testing error. Darker green area represent standard deviation from mean. We can notice some variance between different runs, but all eventually converge to relatively the same optimal.

B. Raw Data, Model, Binning, and Training Set Size

We study the effect of raw data size on Yellow Cab dataset. The training results for YC-1K, YC-10K, YC-100K and YC-1M are in Table III. The testing error becomes smaller when the raw data size increase. To make sure this trend is not a coincidence in sampling, we first created five YC-1K datasets independently, and trained models with the same neural network configuration. The testing RAEs are 3.70%, 3.92%, 4.08%, 4.05%, and 4.30% respectively. This indicates the accuracy stays stable for a given sized dataset and a specific NeuralCubes configuration.

To study this further, we plot histograms of RAEs (the blue bar charts in Table III). The intuition of such histograms is that the RAE of a model over a dataset can be seen as the mean value of the prediction error for each testing query, computed as $error_i = \frac{n|\hat{y}_i - y_i|}{\sum_j |y_j - \bar{y}|}$, where n is the number of testing queries. Now we can simply plot a histogram of *errors*. Note that the y-axis in the histograms are in logarithmic scale. We can clearly see that as the raw data size increase, the model has fewer large errors. We hypothesize that testing error gets smaller as raw data size increase because with more available data, there will be less noise. Since we are using the same neural network configuration, we can expect lower error on datasets that do not require large learning capacity.

TABLE III
YellowCab Dataset with Different Raw Data Size

Name	Raw Data Size	# States	Model Size	Testing RAE
YC-1K	12k (1.8MB)	30k	798KB	3.70%
YC-10K	120k (18MB)	30k	798KB	2.04%
YC-100K	1.2m (180MB)	30k	798KB	1.35%
YC-1M	12m (1.8GB)	30k	798KB	0.97%

To assess the impact on network capacity, we trained a series of networks that increase in size – number of layers and layer width – in order to see how how much prediction improves as network capacity increases. This resulted in the testing four different models with sizes 113KB, 220KB, 798KB, and 1.7MB. The training RAE for these models was 5.06%, 3.93%, 3.59%, and 2.98%, and the testing RAE followed a similar pattern at 5.18%, 4.03%, 3.70%, and 3.10%, respectively. Theses results show that with larger neural networks, we have more capacity in the model and (not surprisingly), the error reduces.

We used the SPLOM dataset to study the effect of binning. When we increase the number of bins, the network requires higher capacity to learn well. As number of bins increased, the model size changed 109KB to 135KB. Testing RAE mainly increased from 1.02% (10 bins), 1.85% (20), 2.25% (30), 2.09% (40), to 2.64% (50). An increase in testing error is to be expected, for several reasons. The first reason is that when the bins are refined, few records fall into the same bin. So the variance within each bin is larger, making it more difficult for NeuralCubes to learn the underlying distribution. The second reason is that when the number of bins increase, the space of possible different queries grows exponentially.

We also tested the same model but with different number of training samples (15k, 30k, and 60k). As we increased the number of samples, the training RAE was 6.74%, 3.59%, and 3.39% and the testing RAE was 6.85%, 3.70%, and 3.47%. We can see a significant accuracy improvement when using 30k training states than 15k training states. However, the improvement from using 30k to 60k is very small, suggesting a possible need to increase the capacity of this neural network.

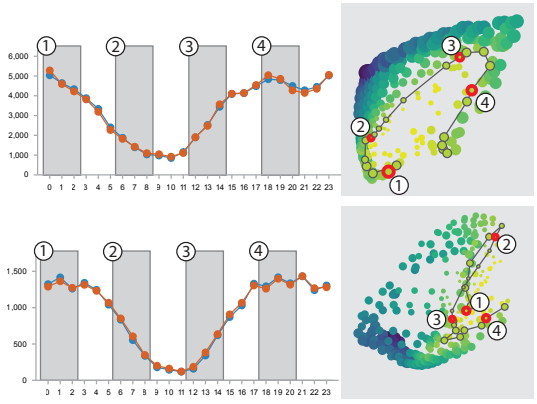

Fig. 10. We compare the diurnal patterns present in the latent space for "hour" in Austin and NYC. The top half shows the queries and latent space for NYC, while the bottom half shows them for Austin. We highlight the sequence of queries in the latent space for sliding ranges throughout the day.

C. Comparing Two Cities by Latent Space

The latent space plots may convey information that could not be directly perceived on histograms of counts. Fig. 10 shows an example. Firstly, the latent space for *hour* of both cities forms a loop for ranges with same length. This suggests that NeuralCubes learned the fact that *hour* indeed has a repeated circular pattern. However, the "circle" in Austin's latent space has a large opening. We find this pattern always exists within multiple independently trained models; though they may have different views (e.g. rotated, stretched) of the latent space due to randomness in the training. This could be caused by the difference of lifestyles of the two cities: New York City never sleeps, while Austin goes to bed at night.

D. Testing with User Queries

In this experiment, we collect queries from real users for evaluation. The models are still trained using artificially generated sample. However, for testing, we setup a web based visualization for users to interact with the four datasets: Brightkite NYC, Brightkite Austin, Yellow Cab and SPLOM dataset. Then we asked two participants to interact with the web UI and log the queries the made. At last, we test the models on these collected queries, which contains more than 100 *states* (40,000+ queries) for each dataset. Compared with testing results using generated queries (Table II), the RAE for user queries are significantly lower, with RAEs of 1.39% (Brightkite NYC), 1.72% (Brightkite Austin), 0.63% (Yellow Cab), and 1.41% (SPLOM). . We hypothesize this improvement happens because humans only selectively explore the query space. However, a full investigation of human behavior in queries is outside the scope of this work.

VIII. Discussion and Limitation

In this work we have presented NeuralCubes: a learning-based approach for computing query aggregations in multi-dimensional datasets. We have shown the accuracy of our model in approximating queries over a wide range of scenarios, and the qualitative benefits of NeuralCubes for downstream use in visual exploration. Although we are encouraged by the results, we recognize several limitations with NeuralCubes that we plan to address in future work.

The main limitation with NeuralCubes is its inability to generalize to novel datasets. The necessity to train a model from scratch, given a new dataset, results in a high computational burden relative to existing datacubes methods [33], [34]. Although generalization, e.g. responding to a query from an arbitrary dataset, is extremely challenging, we believe there is middle ground in reducing the time required for optimization. We will consider efficient optimization schemes for when the data schema is fixed, yet new data items arrive that expand the range of attributes, e.g. check-ins at new points in time for a fixed geographic area. We will further consider meta-learning schemes [15] in order to utilize learned network initializations that exhibit faster convergence, taking advantage of relatedness in queries amongst homogeneous datasets. Additionally, we will extend our architecture to learn over multiple types of queries (e.g. sum, count), leveraging multi-task learning techniques [61].

Another limitation with NeuralCubes is the network design, and its sensitivity to hyperparameter settings for individual datasets – this places an unnecessary burden on the user to make decisions on hyperparameters. For future work, we plan on considering alternative models that are more robust, and lessen the need for hyperparameter tuning. In particular, we plan on investigating attention-based models such as Transformers [14], [55] that can reason over variable-length data queries, e.g. queries over an arbitrary subset of attributes. Whereas our current model simply concatenates attribute-specific feature vectors for learning attribute relationships, we believe self-attention amongst attributes is key for more effective learning, and consequently, simpler network designs with less hyperparameter tuning.

Another limitation with NeuralCubes is the lack of error control. However, we think that it is possible to associate uncertainty with our predictions, extending existing work on approximate Bayesian inference within deep neural networks [16]. Coupled with the use case of visual exploration, we think that this can help alleviate concerns regarding error, by communicating both aggregation results, and confidence in predictions.

One notable feature of NeuralCubes is that the query training set is dependent on the affordances provided by the visual exploration system. This is an advantage in terms of machine learning, because the additional information available allows us to simplify the problem of training a network capable of answering *any* query. Here, NeuralCubes is taking direct advantage of Vapnik's principle, which is "never [...] solve a problem which is more general than the one we actually need to solve." [48]. Nevertheless, we acknowledge that our heuristic for generating queries might not accurately represent what users would query, leading to poor generalization. However, we think that including a small amount of human supervision can provide us with additional information on how to generate

relevant queries for training. Active learning strategies [29] should prove useful in such scenarios to minimize human annotation time.

As a direct comparison to other data cubes based techniques, we built Nanocubes [33] for Yellow Cab dataset (YC-1M) using the same schema as used for NeuralCubes. The size of Nanocubes for YC-1M dataset is 2MB. The size of NeuralCubes models for the same dataset under the same schema is 798KB. This show how NeuralCubes is comparable with Nanocubes in terms of memory usage, but does not provide a detailed query comparison on the same benchmark. Traditional database benchmarks are designed for ad-hoc queries, but the workload of an interactive data exploration system are very different. For example, IDEBench [13] argues that traditional database benchmarks are not suitable for interactive data analysis workload.

The main value of NeuralCubes lies in its speed and extremely small memory footprint in answering queries of a visual data exploration system. Finally, we remark that since the learned model is a differentiable function, NeuralCubes also opens up several opportunities, such as query sensitivity analysis and queries discovery that lead to user-prescribed aggregations.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation (NSF) under grant numbers IIS-1452977, IIS-1513651 and IIS-1815238; and by the Defense Advanced Research Projects Agency (DARPA) under agreement numbers FA8750-17-2-0107 and FA8750-19-C-0002; and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number(s) DE-SC-0019039.

REFERENCES

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 29–42. ACM, 2013.

[2] D. Alabi and E. Wu. Pfunk-h: Approximate query processing using perceptual models. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pp. 1–6, 2016.

[3] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 1363–1375. ACM, 2016.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[5] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2):199–223, 2001.

[6] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on*, pp. 59–66. IEEE, 2008.

[7] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1082–1090. ACM, 2011.

[8] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, pp. 313–328, 2010.

[10] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample+ seek: Approximating aggregates with distribution precision guarantee. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 679–694, 2016.

[11] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[12] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pp. 1538–1546. IEEE, 2015.

[13] P. Eichmann, C. Binnig, T. Kraska, and E. Zgraggen. Idebench: A benchmark for interactive data exploration. *arXiv preprint arXiv:1804.02593*, 2018.

[14] R. Fakoor, P. Chaudhari, J. Mueller, and A. J. Smola. Trade: Transformers for density estimation. *arXiv preprint arXiv:2004.02441*, 2020.

[15] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

[16] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

[17] M. Ghifary, W. Bastiaan Kleijn, M. Zhang, and D. Balduzzi. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the IEEE international conference on computer vision*, pp. 2551–2559, 2015.

[18] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[19] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. *Acm Sigmod Record*, 25(2):205–216, 1996.

[20] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis: The control project. *Computer*, 32(8):51–59, 1999.

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

[22] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[23] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *ACM Transactions on Database Systems (TODS)*, 33(4):1–54, 2008.

[24] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pp. 472–483. IEEE, 2014.

[25] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 547–554. ACM, 2012.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pp. 3581–3589, 2014.

[28] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

[29] A. Kirsch, J. Van Amersfoort, and Y. Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32:7026–7037, 2019.

[30] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 489–504, 2018.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[32] K. Li, Y. Zhang, G. Li, W. Tao, and Y. Yan. Bounded approximate query processing. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2262–2276, 2018.

[33] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[34] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum*, 32(3pt4):421–430, 2013.

[35] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions On Graphics*, 5(2):110–141, 1986.

[36] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE transactions on visualization and computer graphics*, 13(6), 2007.

[37] F. Miranda, H. Doraiswamy, M. Lage, K. Zhao, B. Gonçalves, L. Wilson, M. Hsieh, and C. T. Silva. Urban pulse: Capturing the rhythm of cities. *IEEE transactions on visualization and computer graphics*, 23(1):791–800, 2017.

[38] F. Miranda, L. Lins, J. T. Klosowski, and C. T. Silva. Topkube: a rank-aware data cube for real-time exploration of spatiotemporal data. *IEEE transactions on visualization and computer graphics*, 24(3):1394–1407, 2018.

[39] M. Mitzenmacher. A model for learned bloom filters and related structures. *arXiv preprint arXiv:1802.00884*, 2018.

[40] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pp. 69–84. Springer, 2016.

[41] M. Noroozi, H. Pirsiavash, and P. Favaro. Representation learning by learning to count. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[42] B. of Transportation Statistics. On-time performance. http://www.transtats.bts.gov/Fields.asp?Table ID=236. Accessed: 2018-03-29.

[43] L. Orr, M. Balazinska, and D. Suciu. Entropydb: a probabilistic approach to approximate query processing. *The VLDB Journal*, 29(1):539–567, 2020.

[44] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, pp. 1–4, 2018.

[45] C. A. Pahins, S. A. Stephens, C. Scheidegger, and J. L. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics*, 23(1):671–680, 2017.

[46] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 1461–1476, 2018.

[47] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

[48] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001. doi: 10.1162/089976601750264965

[49] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: Shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 464–475. ACM, 2002.

[50] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[51] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[52] K. Tang, M. Paluri, L. Fei-Fei, R. Fergus, and L. Bourdev. Improving image classification with location context. In *Proceedings of the IEEE international conference on computer vision*, pp. 1008–1016, 2015.

[53] N. Taxi and L. Commission. Yellowcab taxi trip records. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. Accessed: 2018-09-14.

[54] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing for data exploration using deep generative models. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1309–1320. IEEE, 2020.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[56] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[57] Z. Wang, N. Ferreira, Y. Wei, A. S. Bhaskar, and C. Scheidegger. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. *IEEE transactions on visualization and computer graphics*, 23(1):681–690, 2017.

[58] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. *Proceedings of the VLDB Endowment*, 7(10):903–906, 2014.

[59] M. Wu and C. Jermaine. A bayesian method for guessing the extreme values in a data set? In *Proceedings of the 33rd international conference on Very large data bases*, pp. 471–482, 2007.

[60] W. Xu, H. Sun, C. Deng, and Y. Tan. Variational autoencoder for semi-supervised text classification. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[61] Y. Yang and T. M. Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

[62] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-ola: Generalized on-line aggregation for interactive analysis on big data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 913–918, 2015.