Deep Reinforcement-Learning Framework for Exploratory Data Analysis

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Amit Somech
Tel Aviv University
amitsome@mail.tau.ac.il

ABSTRACT

Deep Reinforcement Learning (DRL) is unanimously considered as a breakthrough technology, used in solving a growing number of AI challenges previously considered to be intractable. In this work, we aim to set the ground for employing DRL techniques in the context of Exploratory Data Analysis (EDA), an important yet challenging, that is critical in many application domains. We suggest an end-to-end framework architecture, coupled with an initial implementation of each component. The goal of this short paper is to encourage the exploration of DRL models and techniques for facilitating a full-fledged, autonomous solution for EDA.

CCS CONCEPTS

• **Mathematics of computing → Exploratory data analysis**; • **Theory of computation → Reinforcement learning**;

KEYWORDS

Exploratory Data Analysis, Deep Reinforcement Learning

ACM Reference Format:
Tova Milo and Amit Somech. 2018. Deep Reinforcement-Learning Framework for Exploratory Data Analysis. In *aiDM'18: First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management , June 10, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3211954.3211958

1 INTRODUCTION

Exploratory data analysis (EDA) is an important step in many data driven processes. The purpose of EDA is to better understand the nature of data and to find clues about its properties and quality. EDA is known to be a difficult process, especially for non-expert users, as it requires a deep understanding of the investigated domain as well as the particular context and task at hand [4, 6].

A large body of previous work suggests means for facilitating the EDA process. For example, novel analysis platforms (e.g. Tableau, Kibana, and Splunk) feature simplified interfaces, suitable even for users lacking knowledge in SQL and programing languages. Furthermore, *analysis recommender systems* assist users in formulating queries [1, 6], exploring a data cube [12], and choosing adequate data visualizations [13]. However, EDA is still prominently manual, and seems far from being fully automated.

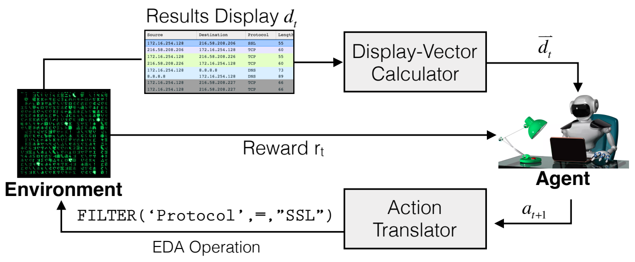

Figure 1: RL-EDA Architecture

Recently, artificial intelligence solutions based on deep neural networks have demonstrated outstanding success, in matching, and even outperforming humans at complex tasks such as playing board games, driving a car, and conducting a natural language dialogue [9]. In particular, many of these solutions employ unsupervised Deep Reinforcement Learning (DRL), where a neural-network based *agent* learns how to perform tasks merely by trial and error, without any human data or guidance (e.g. labeled samples, heuristics). While DRL algorithms successfully perform difficult tasks in a continuously growing applications domain, to our knowledge such solutions have not yet been applied to EDA.

In this work, we make an initial step towards constructing an intelligent, fully autonomous EDA "assistant", backed by a DRL model. The goal of our EDA assistant is to be able to take a new dataset, and automatically "guide" the human analyst through it, by performing a *coherent* sequence of analysis operations that highlight *interesting* aspects of the data. Moreover, it self-learns an optimal policy, by independently interacting with a given dataset.

Typically in a DRL scenario, an autonomous agent is controlled by a deep neural network, and operates within a specific predefined setting, referred to as an *environment*. The environment controls the input that the agent perceives and the actions it can perform: At each time t, the agent observes a *state*, and decides on an *action* to take, according to a self-taught *policy* . The action results in a transition to a new state at time $t + 1$. Additionally, after performing an action the agent obtains a *reward* from the environment. Often, a positive reward is granted for wanted behavior, e.g. winning a game of chess, and negative reward is considered a "punishment", e.g. a driving robot bumping into an obstacle. The goal of the DRL agent is to learn a policy that maximizes the expected utility (i.e. the cumulative reward for its actions).

Employing DRL techniques in the context of EDA is a theoretical and practical challenge: First, one has to devise a machine-compatible representation of the EDA environment, often comprising of a large dataset with values of different types and semantics,

aiDM'18, June 10, 2018, Houston, TX, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5851-4/18/06...$15.00
https://doi.org/10.1145/3211954.3211958

a vast domain of possible analysis actions, and multifaceted result "screens" that may contain features such as grouping and aggregations. Second, at least to our knowledge, there is no clear, explicit reward definition for EDA actions and sessions. Engineering an appropriate reward signal, that ensures that the agent's actions are *interesting*, diverse, and *coherent* to humans is a challenge. Third, determining an adequate network architecture, input and output values, as well as setting its hyper-parameters, are known as difficult tasks, often tailored to the application domain. Determining these for the EDA settings is another challenge.

In this work we suggest an end-to-end DRL-EDA architecture, coupled with an initial design of each component. The goal of this short paper is to encourage the exploration of other approaches and techniques, to facilitate EDA for AI agents. Our contribution can be summarized as follows:

1. We define a generic yet extensible RL environment for EDA, that allows the agent to perform analysis operations (actions) and examine their results (states). At each *state* the agent observes a concise numeric summary of the current results display. To interact with the dataset, we devise a parameterized EDA action space, where embedding techniques ([2, 10]) are employed to allow it to choose dataset values as action parameters.

2. We formulate a compound reward function, which encourages the agent to perform actions that are: (i) interesting - for that we employ a data-driven notion of *interestingness*; (ii) diverse from one another - we use a distance metric between actions' result-sets, and (iii) human understandable - for that we utilize real EDA sessions made by human experts as an exemplar.

3. We sketch an end-to-end design of a DRL agent, utilizing adaptations of state-of-the-art solutions for DRL where the action space is parameterized ([8]), yet large and discrete ([5]).

In what comes next, we recall basic concepts and notations for EDA and RL (Section 2), then describe the architecture and components of our DRL-EDA framework (Section 3). We then overview related work and outline future research directions (Section 4).

2 PRELIMINARIES

To set the ground for our work, we provide common notations from the literature for EDA and for Reinforcement Learning (RL).

The EDA Process. A (human) EDA process starts when a user loads a particular dataset to an analysis UI. The dataset is denoted by $\mathcal{D} = \langle Tup, Attr \rangle$ where Tup is a set of data tuples and $Attr$ is the attributes domain. The user then executes a series of analysis operations (e.g. SQL queries) $q_1, q_2, ..q_n$, s.t. each q_i generates a results *display*, denoted d_i. The results display often contains the chosen subset of tuples and attributes of the examined dataset, and may also contain more complex features (supported by the particular analysis UI) such as grouping and aggregations, results of data mining operations, visualizations, etc.

Reinforcement Learning. As mentioned in Section 1, the RL process describes the operations of an *agent* within an *environment*. Typically, the process is modeled as a *Markov Decision Process* (MDP) that consists of: (1) a set S of all possible states, and (2) a set A of possible actions; (3) A transition *dynamics* $T(s_{t+1}|s_t, a_t)$ that map a state-action pair at time t onto a distribution of states at time $t+1$; (4) a reward function $r_t(s_t, a_t, s_{t+1})$. The agent acts at time t according to a policy $\pi : S \rightarrow p(a|S)$. We use an *episodic* MDP model: For each *episode*, exists an initial state s_0, and the agent reaches a terminate state s_T after performing T actions. The *utility* of an episode is defined as the cumulative (discounted) reward, i.e. $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$ where $\gamma \in (0, 1]$ is a discount factor. The goal is to learn an optimal policy, denoted $\pi^\star$ which achieves the maximum expected utility.

3 DRL-EDA FRAMEWORK

In this section we overview our proposed RL framework for the EDA problem setting. As a proof of concept, we focus on learning within an EDA domain of cyber security, allowing the agent to interact with 4 datasets comprising different network logs, using a limited version of REACT-UI [11] - a modern, web-based EDA interface used for research purposes.

First, we describe the RL environment designed for the context of EDA, representing the EDA "world" in the eyes of the agent (Section 3.1). We then define the reward signal (Section 3.2) and finally describe the design of the agent's deep neural networks (Section 3.3). An illustration of the framework is provided in Figure 1.

3.1 EDA Environment

We developed a simple yet extensible RL environment, in which the actions are EDA operations and the states are their corresponding result sets. In each episode the agent operates on one out of four datasets (selected uniformly at random). In our setting, all datasets share the same schema (comprising 12 attributes), yet their content is different (and independent). At time t, the agent performs a single analysis operation. In general, analysis operations can be expressed using query languages (e.g. SQL), which is difficult for a machine agent to generate from scratch (to date). To simplify, we therefore use *parameterized* analysis operations, that allow the agent to intelligently choose the operation type, then the adequate parameters. Each such operation takes some input parameters and a previous display d (i.e., the results screen of the previous operation), and outputs a corresponding new display. As our work presents a proof of concept, we use a limited set of analysis operations (to be extended in future work). The following actions are supported:

- `FILTER`$(attr, op, term)$ - used to select data tuples that matches a criteria. It takes a column header, a comparison operator (e.g. $=, \geq, contains$) and a numeric/textual term, and results in a new display representing the corresponding data subset.
- `GROUP`$(g\_attr, agg\_func, agg\_attr)$ - groups and aggregates the data. It takes a column to be grouped by, an aggregation function (e.g. SUM, MAX, COUNT, AVG) and another column to employ the aggregation function on.
- `BACK()` - allows the agent to backtrack to a previous display (i.e the results display of the action performed at $t-1$) in order to take an alternative exploration path.

An example `FILTER` operation is given at the bottom of Figure 1.

We next describe how the state observations and actions are encoded for the RL agent.

State Representation. Result displays are often compound, containing both textual and numerical data, that may also be grouped or aggregated. However, a low-level numerical representation of the displays is required to maintain stability and reach learning convergence. In our model, the agent observes at time t a numeric vector, denoted $\vec{d_t}$, that represents the results display d_t obtained after executing an analysis operation at $t-1$. $\vec{d_t}$ contains 50 numerical values representing: (1) 3 structural features for each attribute: its value entropy, number of distinct values, and the number of null values. (2) 1 feature per attribute stating whether it is currently grouped/aggregated, and two global features storing the number of groups and the groups' size variance. At state s_0 the agent observes the statistics of the initial display d_0, representing the dataset instance before any action was taken.

Action Representation. In many RL environments the action space is small and discrete (e.g. in classic control problems, often the agent can choose at time t if to move a step to the left or to the right), which is hardly the case in our setting. As mentioned above, we use a parameterized action space, in which most parameters (attributes, filter operations, aggregation functions) are from a finite discrete domain, but the parameter *term*, representing the value used in the `FILTER` criteria, can be very large. Even when disregarding the filter *term* parameter, the action space is of size 1K.

Our solution incorporates two state-of-the art techniques - we use [8] to deal with the parameterized action space, and [5] to tackle the large discrete space of the *term* parameter.

In accordance with [8], we define the EDA action space as follows. Let A={`FILTER`,`GROUP`,`BACK`} be the set of discrete action types. Each $a \in A$ has a domain of its parameters' values X^a. An action is a tuple (a, x) where $a \in A$ and x is an assignment for the parameters X^a. For example, the EDA operation in Figure 1 is represented by (`FILTER`,{'Protocol',=,"HTTP"}). The action space is thus defined by $\mathcal{A} = \bigcup_{a \in A}\{(a,x) | x \in X^a\}$.

As for representing the *term* parameter, that can theoretically take any numeric/textual value from the domain of the specified attribute, we employ the following solution, inspired by [5]. First, we devise a low-dimensional vector representation for the dataset values, and let the agent at time t generate such a vector. Next, as the generated vector may not correspond to a valid term (i.e. among the discrete action space), we index all term-vectors and perform a *nearest-neighbor lookup* in order to output the valid action closest to the one chosen by the agent.

To obtain a low-level representation of the data *entities* and *tokens* (i.e. all valid column values or a part thereof) we follow [2], and use the well known embedding technique Word2Vec [10]. We handle strings as well as numerical values as suggested in [2]. Performing the lookup for the nearest valid term is done using a fast approximate nearest-neighbor (ANN) index taken from [3].

3.2 Reward Signal Engineering

To our knowledge, there is no clear method for ranking analysis sessions in terms of quality. We therefore design a reward signal for the agent performing IDA sessions, with three goals in mind: (1) Interestingness: Actions inducing *interesting* result-sets should be encouraged. (2) Diversity: actions that yield a display that is similar to the previous ones examined thus far should be penalized. (3) Coherency: actions understandable to humans should induce a higher reward. Correspondingly, our reward signal comprises the following components:

(1) Interestingness. Many measures are suggested in previous work that heuristically capture the interestingness of the output (display) of a given EDA operation. Following [12, 13] we take a *deviation* based approach, in which displays that show different trends compared to the entire dataset are considered more interesting. We adapt the measure suggested in [13], originally for measuring the interestingness of SQL group-by queries with a single aggregated attribute, as follows: Given an action a at time t, we consider its results display, denoted d_t, that can currently be *grouped-by* some column, or not: If d_t is not currently grouped, we consider the value probability distribution under each attribute $attr$ in d_t, denoted P_a. P_a is defined as the relative frequency of the values in $attr$, that sum to 1 (i.e. for each value v of $attr$ in d_t, $p_a(v)$ is the probability to randomly choose v). Then the interestingness of d_t is measured by comparing the value distribution (using, e.g., the KL Divergence measure) of each attribute in d_t to its equivalent in the original dataset $\mathcal{D}$ (i.e., before any action is taken).

For the case that d_t conveys grouping, the interestingness is defined as follow: The grouping and aggregations settings in d_t are employed on the source dataset $\mathcal{D}$, and the deviation (i.e. value distribution distance) is compared only w.r.t. currently aggregated attributes (rather than on all attributes, as above).

(2) Diversity. We want to encourage the agent to choose actions that induce new observations or different parts of the data than those examined thus far. Since multiple sequences of analysis operations may lead to the same results display, we discourage the agent from choosing actions that yield similar result sets to the ones already seen before. To do so, one can compare the similarity between the last observed display d_t to the previous ones. While a handful of diversity definitions exist in previous work, mostly in the domains of web search results and recommendations, we devise a simple yet effective measure, adequate for our context: Since the observation vector $\vec{d_t}$ is a numerical representation of display d_t (Recall from Section 3.1), we define the *diversity reward* to be the sum of the squared Euclidean distances between $\vec{d_t}$ and the vectors of all other displays obtained at time $< t$.

(3) Coherency. To ensure that the agent makes coherent analysis actions, understandable to human analysts, we define a cumulative *coherency reward* that corresponds to the agent's ability to predict actions of human analysts. To estimate this, we use EDA sessions made by expert analysts as an exemplar. For our specific setting, we recorded EDA sessions (actions as well as result displays) made by a group of expert analysts on the datasets used in our environment[1]. The coherencey reward is thus defined as follows: We consider the set of all displays that appeared in the experts' sessions. For each such display, denoted d_u, we calculate the observation vector $\vec{d_u}$ and use it as an input to our DRL agent. We then examine if the particular action (type and parameters) chosen by the agent corresponds to one of the EDA operations taken by one of the human analysts after examining the display d_u Finally, the coherency reward is defined by the count of all matches.

The overall reward for an action is a weighted sum of the components above, where the coherency reward is *cumulative* - given

[1]Since our datasets are from the cyber security domain we recruited participants via dedicated forums, network security firms, and volunteer senior students from the Israeli National Cyber-Security Program.

retroactively to each action after the episode (analysis session) terminates. Additionally, we provide a negative reward for illegal actions (such as grouping by an attribute already grouped, etc.), and a neutral (0) reward for a BACK action - since it is only used to enable the agent to explore different facets of the data that may yield high interestingness scores.

3.3 DRL Agent Design

The agent is designed according to the *actor-critic* approach, a preferable DRL model when the action space is extremely large [7]. In the Actor-Critic model the learning process is decoupled to two different roles: the *actor* performs the action selection, and the *critic* learns a *value function*, from which the action policy is derived. Each of these roles is performed using an independent neural network, as follows: The *actor-network*, denoted μ with a parameters set θ_μ takes as input a state s and outputs an action a in the format of three output vectors (we explain below how these are translated to a single EDA operation). The *critic-network*, denoted Q, with parameters set θ_Q, takes as input a state s and the action vector a and outputs a scalar *Q-Value* ,$Q(s, a)$, which is the expected *utility* (i.e., the cumulative discounted reward) of action a at state s.

Learning Process. For space constraints, we only provide a brief overview of the process. The learning process of the RL agent interweaves the input and output of both networks: At the end of each episode, both networks update their internal parameters w.r.t. loss functions, derived from the utility gained for the actions taken.

The critic-network parameters θ_Q are updated to optimize the Q-function, by minimizing a *loss function* derived from the actual utility gained at the end of each episode. Then, the critic-network produces gradients that indicate how the actor's actions should be changed in order to increase the Q-Value. The actor uses these gradients to update its parameters θ_μ, by minimizing a loss function, determined by the cumulative distance between the actor's action and the approximated "optimal" actions.

Actions to EDA Operations. At each state t, the agent passes the actor-network's output action to the environment. The output action comprises three parts: (1) A numeric score for each action type. (2) A numeric score for each discrete parameter. (3) A continuous *term vector* (recall from Section 3.1). The corresponding EDA operation is obtained as follows: First, the operation type is determined according to the score in (1) that obtained the highest value. Then, the corresponding discrete parameters (e.g. *attr, agg_func*) are determined from (2) in a similar manner. Last, if the chosen action type is FILTER - a the output term vector is translated to a valid term, using the nearest-neighbor lookup, as explained in Section 3.1.

4 CHALLENGES & RELATED WORK

A battery of tools have been developed over the last years to assist analysts in interactive data exploration[4, 6, 13], by e.g. suggesting adequate visualizations [13] and SQL/OLAP query recommendations [1, 6]. Particularly, [4] presents a system that iteratively presents the user with interesting samples of the dataset, based on manual annotations of the tuples. As opposed to previous work, we propose a self-learning agent, capable of intelligently performing a *sequence* of EDA operations on a given dataset.

However, our framework is merely an initial step towards facilitating a full-fledged, machine-learning based solution for EDA. Future challenges and questions are as follows.

Evaluation and benchmark. As the implementation of our framework is still a work in progress, an experimental evaluation is called for. This raises the question of how one can devise benchmark tests, evaluating the interestingness, comprehensiveness, and coherency of machine-generated analysis sessions.

Broad analysis capabilities. Supporting more types of analysis actions (e.g. visualizations, data mining), and the ability to analyze datasets of different schemata and for different purposes is a challenging future research.

Optimized representations. Recent techniques for *embedding-based representation learning* (e.g. [10]) obtain tremendous success in finding low-level representation for compound elements (e.g. words, graph nodes, multimedia images), such techniques can be used to investigate how one may find more comprehensive, yet low-dimensional, representations in the context of EDA: e.g., for analysis operations, dataset values, and entire displays.

Applications and use cases. The application possibilities for an autonomous EDA agent are endless, since EDA is ubiquitous in almost every domain. For example, combining it with state-of-the-art *question-answering* systems may vastly expand the range of questions that such agents can answer. Additionally, using the EDA agent in the context of *fraud-detection* and *cyber-security threat analysis* may also be of great importance.

ACKNOWLEDGMENTS

This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by grants from Intel, the Israel Innovation Authority, and the Israel Science Foundation.

REFERENCES

[1] J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, and S. Rizzi. A collaborative filtering approach for recommending olap sessions. *ICDSST*, 2015.
[2] R. Bordawekar, B. Bandyopadhyay, and O. Shmueli. Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities. *arXiv:1712.07199*, 2017.
[3] L. Boytsov and B. Naidan. Engineering efficient and effective non-metric space library. In *SISAP*, 2013.
[4] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Aide: An active learning-based approach for interactive data exploration. *TKDE*, 2016.
[5] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv:1512.07679*, 2015.
[6] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *TKDE*, 2014.
[7] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE SMC*, 2012.
[8] M. Hausknecht and P. Stone. Deep reinforcement learning in parameterized action space. *arXiv:1511.04143*, 2015.
[9] Y. Li. Deep reinforcement learning: An overview. *arXiv:1701.07274*, 2017.
[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.
[11] T. Milo and A. Somech. React: Context-sensitive recommendations for data analysis. In *SIGMOD*, 2016.
[12] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.
[13] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *VLDB*, 2015.