3 Lessons Learned from Implementing a Deep Reinforcement Learning Framework for Data Exploration

Ori Bar El, Tova Milo, and Amit Somech
Tel Aviv University, Israel

ABSTRACT

We examine the opportunities and the challenges that stem from implementing a Deep Reinforcement Learning (DRL) framework for Exploratory Data Analysis (EDA). We have dedicated a considerable effort in the design and the development of a DRL system that can autonomously explore a given dataset, by performing an entire *sequence* of analysis operations that highlight *interesting* aspects of the data.

In this work, we describe our system design and development process, particularly delving into the major challenges we encountered and eventually overcame. We focus on three important lessons we learned, one for each principal component of the system: (1) Designing a DRL *environment* for EDA, comprising a machine-readable encoding for analysis operations and result-sets, (2) formulating a reward mechanism for exploratory sessions, then further tuning it to elicit a desired output, and (3) Designing an efficient neural network architecture, capable of effectively choosing between hundreds of thousands of distinct analysis operations.

We believe that the lessons we learned may be useful for the databases community members making their first steps in applying DRL techniques to their problem domains.

1. INTRODUCTION

Exploratory Data Analysis (EDA) is an important procedure in any data-driven discovery process. It is ubiquitously performed by data scientists and analysts in order to understand the nature of their datasets and to find clues about their properties, underlying patterns, and overall quality.

EDA is known to be a difficult process, especially for non-expert users, since it requires profound analytical skills and familiarity with the data domain. Hence, multiple lines of previous work are aimed at facilitating the EDA process [5, 14, 17, 3], suggesting solutions such as simplified EDA interfaces for non-programmers (e.g., Tableau[1], Splunk[2]), and analysis recommender-systems that assist users in formulating queries [5, 14] and in choosing data visualizations [17]. Still, EDA is predominantly a manual, non-trivial process that requires the undivided attention of the engaging user.

In recent years, artificial intelligence systems based on a Deep Reinforcement Learning (DRL) paradigm have surpassed human capabilities in a growing number of complex tasks, such as playing sophisticated board games, autonomous driving, and more [10]. Typically in such solutions, an artificial 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. After performing an action, the agent obtains a positive or negative *reward* from the environment, either to encourage a successful move or discourage unwanted behavior.

In this work, we examine the opportunities and the challenges that stem from implementing a DRL framework for data exploration. We have dedicated a considerable effort in the design and the development of a DRL system that can autonomously explore a given dataset, by performing an entire *sequence* of analysis operations that highlight *interesting* aspects of the data. Since it uses a DRL architecture, our system learns to perform meaningful EDA operations by independently interacting with multiple datasets, without any human assistance or supervision.

At first sight, the idea of applying DRL techniques in the context of EDA seems highly beneficial. For instance, as opposed to current solutions for EDA assistance/recommendations that are often heavily based on users' past activity [5, 14] or real-time feedback [3], a DRL-based solution has no such requirements since it trains merely from self-interactions. Also, since its training process is performed offline, a DRL-based system may be significantly more efficient in terms of running times, compared to current solutions that compute recommendations at interaction time.

However, employing a DRL architecture for EDA also poses highly non-trivial obstacles that we tackled throughout our development process:

(1) EDA Environment Design: What information to include and what to exclude? Since (to our knowledge) DRL solutions have not yet been applied to EDA, our first challenge was to design an *EDA environment*, in which an artificial agent can explore a dataset. The environment is a critical component in the DRL architecture as it controls what the agent can "see" and "do". In the con-

[1]https://www.tableau.com
[2]https://www.splunk.com

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and AIDB 2019.
1st International Workshop on Applied AI for Database Systems and Applications (AIDB'19), August 26, 2019, Los Angeles, California, CA, USA.

text of EDA, the agent can "do" analysis operations (e.g. filter, group, aggregations) and "see" their result sets. However, in EDA, datasets are often large and comprise values of different types and semantics. Also, EDA interfaces support a vast domain of analysis operations with compound result sets, containing layers such as grouping and aggregations. Correspondingly, it is particularly challenging to design a machine-readable representation for analysis operations and result sets, that facilitates an efficient learning process. For example, including too little information in the results-encoding may not be informative enough for the agent to make "correct" decisions, thereby hinder the learning convergence. On the other hand, including too much information may negatively effect the generalization power of the model, and encourage overfitting.

(2) Formulate a reward system for EDA operations. Another crucial component in any learning based system is an explicit and effective reward function, which is used in the optimization process of the system. As opposed to most existing DRL scenarios (such as board games, video games), to our knowledge, there is no such explicit reward definition for EDA operations. Ideally, we want the agent to perform a sequence of analysis operations that are both (i) interesting, (ii) diverse from one another, and (iii) coherent, i.e., human understandable. The challenge in formulating a new reward signal is twofold: first, to properly design and implement the reward components and achieve a positive, steady learning curve. Second, even after successfully implementing the reward components, the agent still demonstrated unwanted behavior. Therefore, one has to further analyze the reward mechanism and learning process, and derive the appropriate adjustments.

(3) Design a deep network architecture that can handle thousands of different EDA Operations. Typically in Deep Reinforcement Learning (DRL), at each state the agent chooses from a finite, small set of possible actions. However, even in our simplified EDA environment there are over $100K$ possible distinct actions. Experimenting first with off-the-shelf DRL architectures (such as DQN and A3C [10]) that assume a small set of possible actions, we observed that the learning process does not converge. Also, applying dedicated solutions from the literature (e.g., [6, 4]) resulted in unstable and ineffective learning. Therefore, the challenge here is to utilize the structure of the action-space in designing a novel network architecture that is able to produce a successful, converging learning process.

A short paper describing our initial system design was recently published in [13]. In this work, we revisit that initial design, contemplating on the ideas that indeed worked in practice and the ideas that were abandoned or modified. We believe that the lessons we learned during the development process may be useful for the databases community members making their first steps in applying DRL techniques to their problem domains.

Paper Outline. We start by recalling basic concepts and notations for EDA and DRL (Section 2). Then, in Section 3 we examine our development process and provide insights regarding each of the "lessons" we learned: EDA environment design (Section 3.1), Reward Signal Formulation (Section 3.2), and Neural-Network Construction (Section 3.3). Last, we conclude and review related work in Section 4.

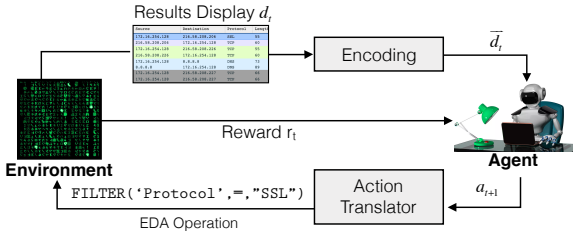

Figure 1: DRL Environment for EDA

2. TECHNICAL BACKGROUND

We recall basic concepts and notations for EDA and DRL.

The EDA Process. A (human) EDA process begins 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 $q_1, q_2, ..q_n$, 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.

Typically, DRL is concerned with an agent interacting with an environment. The process is often modeled as a Markov Decision Process (MDP), in which the agent transits between state by performing actions. At each step, the agent obtains an observation from the environment on its current state, then it is required to choose an action. According to the chosen action, the agent is granted a reward from the environment, then transits to a new state. We particularly use an episodic MDP model: For each episode, the agent starts at some initial state s0, then it continues to perform actions until reaching a terminus state. The utility of an episode is defined as the cumulative reward obtained for each action in the episode. The goal of a DRL agent is learning how to achieve the maximum expected utility.

3. LESSONS FROM DEVELOPING A DRL SYSTEM FOR EDA

We describe our system development process, particularly delving into the major obstacles and challenges we encountered and eventually overcame. Each lesson summarizes our insights regarding a main component of the DRL system.

3.1 Lesson #1: DRL Environment Design

The first challenge we encountered in developing a DRL system was to design a computerized environment for EDA.

The principal idea, as we also described in [13], was to define the environment's *action-space* as the set of allowed EDA operations, and its *state-space* as the overall set of possible result-displays. The environment contains a collection of datasets - all sharing the same schema, yet their instances are different (and independent). In each episode (i.e., EDA session) of length N, the agent is given a dataset $\mathcal{D}$, chosen uniformly at random, and is required to perform N consecutive EDA operations. Figure 1 provides a high-level illustration of the proposed DRL-EDA environment.

The crux of environment design, from our perspective, is twofold: (1) How to represent and control what the agent can "do"? For instance, should we allow it an expressive,

flexible interface such as free-form SQL? (2) How to properly encode what the agent is "seeing"? Namely, how to devise a machine-readable representation of result-displays, that are often large and complex?

How to define the EDA action-space. Our initial idea for EDA operations representation, was to simply use an established query language for structured data (e.g., SQL, MDX), mainly since these languages are highly expressive, and frequently used in both research and industry for the past several decades. However, generating structured queries is a known difficult problem, currently in the spotlight of active research areas such as *question answering over structural data* [18] and *natural language database-interfaces* [8]. In both these domains, existing works rely on (1) the existence of a sufficiently large annotated queries repository, and (2) the fact that useful information (such as the *WHERE* clause) can be extracted from the natural-language input question. In the context of EDA, both these requirements are irrelevant, as the system is expected to generate queries without any human reference.

Correspondingly, our EDA environment supports *parameterized* EDA operations, allowing the agent to first 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 results display. In our prototype implementation, we use a limited set of analysis operations (to be extended in future work):

`FILTER`$(attr, op, term)$ - used to select data tuples that match 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 (An example `FILTER` operation is given at the bottom of Figure 1).

`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.

While complex queries (comprising joins, sub-queries, etc.) are not yet supported, the advantages of our simple action-space design are that (1) actions are atomic and relatively easy to compose (e.g., there are no syntax difficulties). (2) queries are formed *gradually* (e.g., first employ a `FILTER` operation, then a `GROUP` by some column, then aggregate by another, etc.), as opposed to SQL queries where the entire query is composed "at once". The latter allows fine-grained control over the system's output, since each atomic action obtains its own *reward* (See Section 3.2).

Nevertheless, even in our simplified EDA environment the size of the action space reaches hundreds of thousands of actions, which poses a crucial problem for existing DRL models. We explain how we confronted this issue in Section 3.3.

How to define the environment's states representation. The agent decides which action to perform next mostly based on the *observation-vector* it obtains from the environment at each state. Therefore, the information, as well as the way it is encoded in the observation-vector, is of high importance. Intuitively, the observation should primarily represent the results display of the last EDA operation performed by the agent. However, result displays are often compound, containing both textual and numerical data which may also be grouped or aggregated. Therefore, the result displays cannot be passed to the agent "as-is". The main challenges in designing the observation-vector are thus (i) to devise a uniform, machine-readable representation for results-displays and (ii) to identify what information is necessary for the agent to maintain stability and reach learning convergence.

i. Result-displays representation. We devised a uniform vector representation for each results display, representing a compact, structural summary of the results. It comprises: (1) three descriptive features for each attribute: its values' entropy, number of distinct values, and the number of null values. (2) one feature per attribute stating whether it is currently grouped/aggregated, and three global features storing the number of groups and the groups' size mean and variance.

While this representation ignores the semantics of a results-display (as it contains only a structural summary), a similar approach was taken in an EDA next-step recommender system [14] developed by a subset of the authors of this work. It is empirically demonstrated in [14] that such representation of result displays is useful for predicting the next-step in an EDA session, and also for *transfer-learning*, i.e., better utilization of EDA operations performed over *different datasets* (exploiting structurally similar displays).

ii. Include session information. Indeed, when using just the encoded vector of the last results-display, our prototype implementation reaches learning convergence (i.e., maximizing the cumulative reward as described in Section 3.2). The orange line in Figure 2 depicts the learning curve of the agent when using a single encoded results-display as an observation vector. However, see that the learning process is rather slow and fluctuating, which may imply that the information encoded in the observation is insufficient for the agent to obtain a steady learning rate. Now, the question is

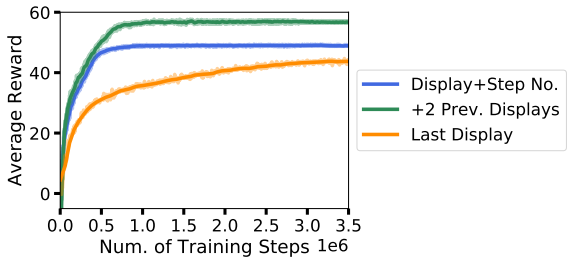

Figure 2

what *additional* information should be encoded in the observation? Intuitively, if the agent is required to perform a *sequence* of operations, it may be useful to encode information about the entire session, rather than just the current display. However, encoding too much information may slow down and even hinder the learning process.

We attempted two approaches for including session information: First, we tried to include the agent's current *step-number* (using one-hot encoding). This is a rather small yet informative addition to the observation. The blue line in Figure 2 depicts the learning curve when using this approach. At first sight it may seem that adding the step number to the observation is useful, as the blue learning curve converges much faster and to a higher value than the

orange one (describing the single-display observation). However, when further analyzing this approach - we noticed that regardless of the given dataset, the output EDA operations sequence hardly varied. This means that it overfits the step-number, ignoring the rest of the information provided.

Our third (and successful) idea was to form a more elaborate observation that includes, in addition to the current display vector, the vectors of the last two previous displays (Here also, a similar approach was taken in our EDA next-step recommender system [14] and was proven useful). While this approach triples the size of the original observation vector, the convergence of the learning curve (see the green line in Figure 2) is faster than the first approach (single display), much more stable and reaches the highest average reward.

Lesson #1 - insights summary: *(1) limiting the environment's supported actions to simple, atomic operations allows for a controllable, easier to debug DRL environment. (2) The kind of information encoded in the observation vector is critical to obtain a converging learning curve.*

3.2 Lesson #2: Reward Signal Development

Since EDA is a complex task, designing an effective reward mechanism that elicit desired behavior is quite a challenge. In the absence of an explicit, known method for ranking analysis sessions, we developed a reward signal for EDA actions with three goals in mind: (1) Actions inducing *interesting* result sets should be encouraged. (2) Actions in the same session should yield *diverse* results describing different parts of the examined dataset, and (3) the actions should be *coherent*, i.e. understandable to humans.

We next discuss two major obstacles that we tackled: (i) effectively implementing the reward signal's components, and (ii) further tuning the reward signal to effectively encourage desired behavior.

Reward Signal Implementation. The cumulative reward is defined as the weighted sum of the following individual components. The first two components, *interestingness* and *diversity* were rather straightforward to implement. It was particularly challenging to develop the *coherency* reward.

(1) interestingness. To rank the interestingness of a given results-display we use existing methods from the literature. We employ the *Compaction-Gain* [2] method to rank `GROUP` actions (which favors actions yielding a small number of groups that cover a large number of tuples). To rank `FILTER` actions we use a relative, deviation-based measure (following [17]) that favors actions' results that demonstrate significantly different trends compared to the entire dataset.

(2) Diversity. We use a simple method to encourage the agent to choose actions inducing new observations of different parts of the data than those examined thus far: We calculate the Euclidean distances between the display-vector $\vec{d_t}$ (representing the current results display d_t) and the vectors of all previous displays obtained at time $< t$.

(3) Coherency. Encouraging *coherent* actions is a rather unique task in the field of DRL. For example, when playing a board game such as chess or Go, the artificial agent's objective is solely to win the game, rather than performing moves that make sense to human players. Yet, in the case of EDA, the sequence of operations performed by the agent *must* be understandable to the user, and easy to follow. We first briefly explain our original implementation and the reason it failed, then explain the changes that we made to develop a working solution.

Our initial idea for implementing a coherency reward was to utilize EDA sessions made by expert analysts as an exemplar (We already had a collection of relevant exploratory sessions, from the development of [14]). Hence, we devised an auxiliary test to evaluate the agent's ability to *predict actions of human analysts*. Intuitively, if the agent performs similar EDA operations to the ones employed by human users at the same point of their analysis sessions - then the agent's actions are coherent. The coherency test was performed after each training batch, then a *delayed* reward, corresponding to the coherency score obtained in the test, was granted uniformly to all actions in the following episodes.

The orange line in Figure 3 depicts the learning curve, particularly for the prediction-based coherency reward. See that the obtained coherency reward remains close to 0 even at the end of the training process. We believe that the failure to learn stems from two reasons: first, the states of the human sessions examined in the auxiliary test were often unfamiliar states that the agent did not encounter during training. Second, the coherency reward was divided uniformly over all actions, hence the learning agent was not able to "understand" which particular actions contribute more to the coherency reward, and which do not.

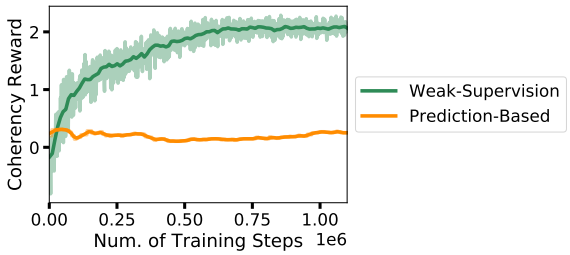

Figure 3

We then developed a second (successful) coherency signal, based on *weak-supervision*. Learning from the flaws of the prediction-based solution, we built a classifier for ranking the degree of coherency of *each action* (rather than provide an overall score, distributed to all actions uniformly). However, since a training dataset containing annotated EDA actions does not exist, we employed a weak-supervision based solution. Based on our collection of experts' sessions, we composed a set of heuristic classification-rules (e.g. *"a group-by employed on more than four attributes is non-coherent"*), then employed Snorkel [15] to build a weak-supervision based classifier that lifts these heuristic rules to predict the coherency level of a given EDA operation. The coherency classifier is then used to predict the coherency-level of each action in the agent's session, and grants it a corresponding reward. The green line in Figure 3 depicts the learning curve w.r.t. the weak-supervision coherency reward. Indeed, this time the learning process steadily converges.

Tuning the reward signal. Using the combined reward signal described above, our model achieved a positive, converging learning curve for each component. However, when inspecting the outputted sequences of EDA operations the results were still not satisfying, i.e., the agent displayed unwanted behavior. For example, we noticed the two following

issues: (i) the agent largely prefers to employ GROUP operations and hardly performs FILTER operations. (ii) The first few EDA operations in each session were considerately more suitable, compared to the later actions in the same session.

To understand the origin of such behavior, we performed an extensive analysis of the reward signal and learning process. We discovered, indeed, that both these issues stem from the reward signal distribution, and can be easily corrected. As for the first issue, Figure 4 shows the cumulative reward granted for each action type (green bars), in comparison to the proportional amount it was employed by the agent (blue bars). Interestingly, GROUP operations are, on

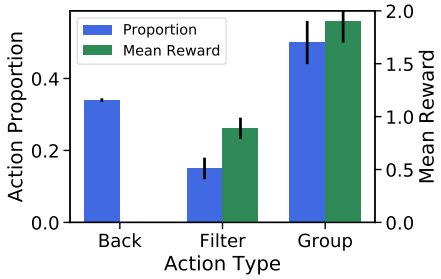

Figure 4

average, more rewarding than FILTER operations, which explains the agent's bias towards GROUP operations. Examining the second issue, Figure 5 depicts the averaged reward obtained at each step in a session (with a translucent error band). It is visibly clear that the first few steps obtain a much larger reward than the later ones.

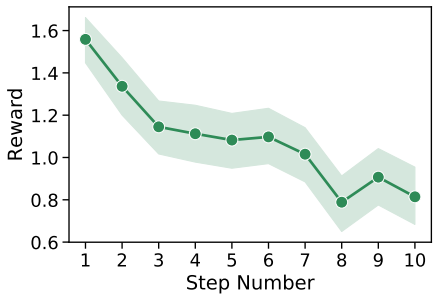

Figure 5

To overcome both issues, we corrected the reward signal by (i) modifying the cumulative signal by adding more weight to FILTER actions, and (ii) adding a monotone decreasing coefficient to the signal, w.r.t. the step number.

Lesson #2 - insights summary: *When designing a reward mechanism from scratch, one has to first make sure that a positive learning curve can be obtained with the developed signal. Once this is done, it is also required to analyze the agent's behavior, reward distribution and learning process, then adjust the signal to elicit desired behavior.*

3.3 Lesson #3: Network Architecture Design

As oppose to most DRL settings, in our EDA environment the action-space is parameterized, very large, and discrete. Therefore, directly employing off-the-shelf DRL architectures is extremely inefficient since each distinct possible action is often represented as a dedicated node in the output layer (see, e.g. [4, 10]).

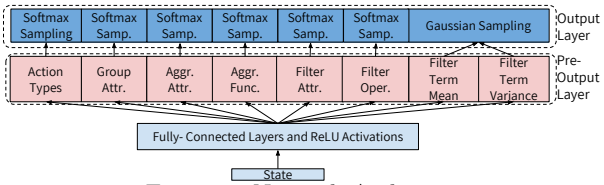

Figure 6: Network Architecture

Our first architecture was based on the adaptation of two designated solutions from the literature ([6, 4]). While this approach did not work as desired, after analyzing its performance we devised a second, successful architecture based on a novel multi-softmax solution. We next briefly outline both architectures.[3]

Architecture 1: Forced-Continuous. Briefly, [6] suggests an architecture for cases in which the actions are parameterized yet *continuous*. Rather than having a dedicated node per distinct action - the output layer in [6] comprises a node for each action type, and a node for each parameter. While this approach dramatically decreases the network's size, the output of each node is a *continuous* value, which is not the case in our EDA environment (the parameters have a *discrete* values domain). Therefore, to apply this approach in our context we formed a continuous space for each discrete parameter, by dividing the continuous space to equal segments, one for each discrete value. Then, to handle the value selection for the *term* parameter of the FILTER operation, that can theoretically take any numeric/textual value from the domain of the specified attribute, we followed [4] which tackles the action selection from a large yet discrete space. The authors suggest first devising a low-dimensional, continuous vector representation for the discrete values (the dataset tokens, in our case), then letting the agent generate such a vector as part of its output. Encoding the dataset tokens was done following [1] using an adaptation of Word2Vec [12].

The blue line in Figure 7 depicts the learning curve when using the solution mentioned above. While the convergence rate is unstable, it eventually reaches a rather high reward. However, its main drawback is that when performing a random shuffle in the way the values are discretized (e.g., shuffle the attributes' order) - a much lower reward is obtained (as depicted by the orange line in Figure 7). Namely, the performance of this model is greatly affected by the particular discretization of the continuous parameters space.

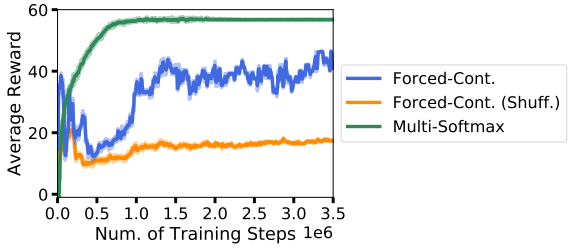

Figure 7

Architecture 2: Multi-Softmax. Our novel architecture utilizes the parametric nature of the action space, and allows the agent to choose an action type and a value for each parameter. This design reduces the size of the output layer to (approximately) the cumulative size of the parameters' value domains. While the output layer is larger than that of

[3]Both are based on the Actor-Critic paradigm (See [10]).

Architecture 1, it is still significantly smaller than in the off-the-shelf solutions, where each parameters' instantiation is represented by a designated node. Architecture 2 is depicted in Figure 6. Briefly, we use a "pre-output" layer, containing a node for each action type, and a node for each of the parameters' values. Then, by employing a "multi-softmax" layer, we generate separate probability distributions, one for action types and one for each parameter's values. Finally, the action selection is done according to the latter probability distributions, by first sampling from the distribution of the action types ($a \in A$), then by sampling the values for each of its associated parameters.

Then, to handle the "term" parameter selection, we utilize a simple solution to map individual dataset tokens to a single yet *continuous* parameter. The continuous term-parameter, computed ad-hoc at each state, represents the *frequency of appearances* of the dataset tokens in the current results-display. Finally, instantiating this parameter is done merely with two entries in our "pre-output" layer: a mean and a variance of a Gaussian (See Figure 6). A numeric value is then sampled according to this Gaussian, and translated back to an actual dataset token by taking the one having the closest frequency of appearance to the value generated by the network.

The green line in Figure 7 depicts the learning curve when using Architecture 2. Indeed, it converges much faster than Architecture 1, obtains a higher reward on average and, most importantly, it is not depended on a particular order of the parameters' values.

Lesson #3 - insights summary: *Handling a DRL environment with a large, discrete action space is a non-trivial challenge. In our case, we utilized the parametrized nature of the actions to design an effective network architecture.*

4. CONCLUSION & RELATED WORK

A battery of tools have been developed over the last years to assist analysts in data exploration [7, 5, 17, 3, 14], by e.g. suggesting adequate visualizations [17] and SQL query recommendations [5]. Particularly, [3] presents a system that iteratively presents the user with interesting samples of the dataset, based on manual annotations of the tuples. Different from these solutions, our DRL based system for EDA is capable of self-learning how to intelligently perform a *sequence* of EDA operations on a given dataset, solely by autonomous self-interacting.

DRL is unanimously considered a breakthrough technology, with a continuously growing number of applications and use cases [10]. While it is not yet widely adopted in the databases research community, some recent works show the incredible potential of DRL in the context of database applications. Interestingly, while these works present solutions for different problem domains, inapplicable to EDA, they mention some similar DRL-related difficulties to the ones described in our work. For example, [9] describes a DRL-based scheduling system for *distributed stream data processing*. Although work scheduling and EDA are completely different tasks, similar DRL challenges are tackled in [9], e.g., designing a machine-readable encoding for the states (in their case, describing the current workload and scheduling settings), and handling a large number of possible actions (assignment of tasks to machines). Additionally, [16] and [11] present prototype systems for *join-order optimization* for RDBMS. These two short papers also encounter DRL-related challenges, such as designing a state-representation (that can effectively encode join-trees and predicates), formulate a reward signal (based on query execution cost models), and more. We therefore believe that the lessons and insights obtained throughout our system development process may be useful not only for EDA system developers yet to many more database researchers experimenting with DRL to solve other databases problems.

Acknowledgements. This work has been partially funded by the Israel Innovation Authority, the Israel Science Foundation, Len Blavatnik and the Blavatnik Family foundation, and Intel® AI DevCloud.

5. REFERENCES

[1] R. Bordawekar, B. Bandyopadhyay, and O. Shmueli. Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities. *arXiv preprint arXiv:1712.07199*, 2017.

[2] V. Chandola and V. Kumar. Summarization - compressing data into an informative representation. *KAIS*, 12(3), 2007.

[3] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Aide: An active learning-based approach for interactive data exploration. *TKDE*, 2016.

[4] 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 preprint arXiv:1512.07679*, 2015.

[5] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *TKDE*, 2014.

[6] M. Hausknecht and P. Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[7] R. E. Hoyt, D. Snider, C. Thompson, and S. Mantravadi. Ibm watson analytics: automating visualization, descriptive, and predictive statistics. *JPH*, 2(2), 2016.

[8] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1), 2014.

[9] T. Li, Z. Xu, J. Tang, and Y. Wang. Model-free control for distributed stream data processing using deep reinforcement learning. *PVLDB*, 11(6), 2018.

[10] Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[11] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *aiDM*, 2018.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[13] T. Milo and A. Somech. Deep reinforcement-learning framework for exploratory data analysis. In *aiDM*, 2018.

[14] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *KDD*, 2018.

[15] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3), 2017.

[16] I. Trummer, S. Moseley, D. Maram, S. Jo, and J. Antonakakis. Skinnerdb: regret-bounded query evaluation via reinforcement learning. *PVLDB*, 11(12), 2018.

[17] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13), 2015.

[18] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.