Automatic Data Acquisition for Deep Learning

Jiabin Liu[1], Fu Zhu[1], Chengliang Chai[1], Yuyu Luo[1], Nan Tang[2]
[1]Tsinghua University, China; [2]QCRI, Qatar
{liujb19@mails.,zhuf18@mails.,chaicl15@mails.,luoyy18@mails.}tsinghua.edu.cn,ntang@hbku.edu.qa

ABSTRACT

Deep learning (DL) has widespread applications and has revolutionized many industries. Although automated machine learning (AutoML) can help us away from coding for DL models, the acquisition of *lots of* high-quality data for model training remains a main bottleneck for many DL projects, simply because it requires high human cost. Despite many works on weak supervision (*i.e.,* adding weak labels to seen data) and data augmentation (*i.e.,* generating more data based on seen data), automatically acquiring training data, via smartly searching a pool of training data collected from open ML benchmarks and data markets, is not explored.

In this demonstration, we demonstrate a new system, automatic data acquisition (AutoData), which automatically searches training data from a heterogeneous data repository and interacts with AutoML. It faces two main challenges. (1) How to search high-quality data from a large repository for a given DL task? (2) How does AutoData interact with AutoML to guide the search? To address these challenges, we propose a reinforcement learning (RL)-based framework in AutoData to guide the iterative search process. AutoData encodes current training data and feedbacks of AutoML, learns a policy to search fresh data, and trains in iterations. We demonstrate with two real-life scenarios, image classification and relational data prediction, showing that AutoData can select high-quality data to improve the model.

PVLDB Reference Format:
Jiabin Liu[1], Fu Zhu[1], Chengliang Chai[1], Yuyu Luo[1], Nan Tang[2]. Automatic Data Acquisition for Deep Learning. PVLDB, 14(12): XXX-XXX, 2021.
doi:10.14778/3476311.3476333

1 INTRODUCTION

Deep learning (DL) is gaining much popularity due to its powerful and mysterious in terms of accuracy and generalization ability, which has widespread applications, *e.g.,* image recognition [7], natural language processing [13, 16], advertisement recommendation [8], etc. To democratize DL, automated machine learning (AutoML) is proposed as a promising solution to build a DL system without human assistance and has made great strides. Nevertheless, even with the help of AutoML, lots of labeled training data is still a must. One obstacle is that getting access to enough high-quality training data is usually both time-consuming and labor-intensive. Therefore, a challenging problem is how to acquire training data automatically, so as to improve a downstream DL task.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476333

*Chengliang Chai is the corresponding author.

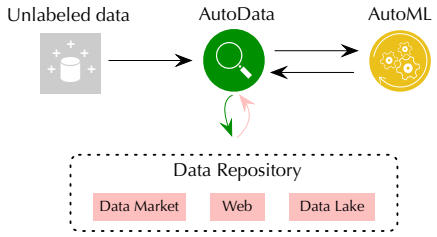

Figure 1: Overview of AutoData

Opportunity. In the age of big data, there is plenty of high-quality labeled data available in external data repositories. For example, there are 3000+ datasets on www.paperswithcode.com alone, where a big dataset can have 14M+ labeled data points. Moreover, it is also possible to acquire data from data markets.

Challenges. Automatic data acquisition faces two main challenges. (1) How to search valuable data for the downstream task? Since the data in the wild is heterogeneous, not all of them help [1, 10], and some may even degrade the model performance *w.r.t.* a given task. On the other hand, good training data may come from multiple sources [2, 4, 5, 12], so intuitively, the fundamental question is: how to search data points that are useful to a DL task from a large number of heterogeneous data? (2) How to interact with AutoML when searching data? The data search process cannot work without interacting with AutoML. Although there are many metrics to measure the overall performance of the dataset on the model, what information do we need from AutoML to guide the search?

Our Proposal: RL-based AutoData ↻ AutoML. To address these challenges, we propose a reinforcement learning (RL)-based automatic data search system AutoData, which fetches fresh training data from heterogeneous repositories and interacts with AutoML, as shown in Figure 1.

RL is an important ML paradigm that an "agent" learns from the feedback through trial-and-error interactions with the "environment". Specifically, given a training dataset fed into AutoML, the environment in our RL framework has a *Valuator* to compute an *influence score* for each training data point, which measures how the model changes if the point is modified a little. The higher the score is, the larger impact it has on the model. Therefore, the scores together with the data points serve as the "state" in RL. Given the state, the *Search-Policy* module in the agent selects the optimal action (*i.e.,* fetch a batch of fresh training data), and then feeds them into AutoML again. Then the feedback of AutoML is used as "reward" to learn the *Search-Policy*.

Demonstration Scenarios. AutoData works for multiple data types. We will show two popular cases. *(1) Image Classification.* AutoData can discover images from the external repositories

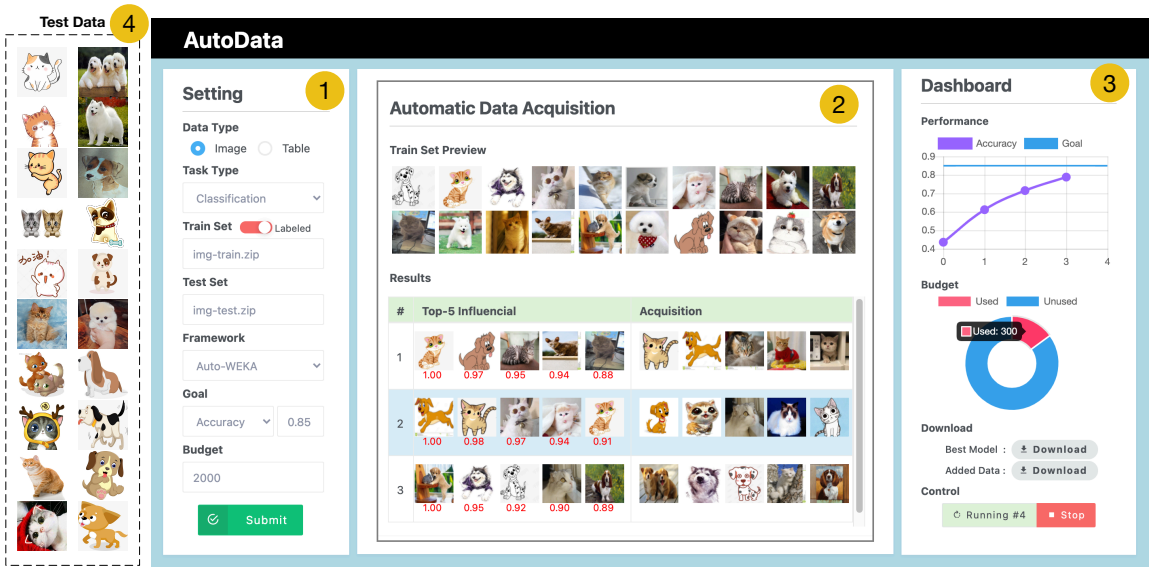

Figure 2: A Running Example of AutoData

(*e.g.,* images crawled from Google, Baidu or ImageNet [7]) to enrich the training set. We will demonstrate how the user sends requests to AutoData and show the discovering process of image data as well as the performance improvement in iterations. *(2) Relational Data Prediction.* Our data repository includes data lakes like NYU Auctus [6], where AutoData interacts with them through APIs. For instance, given the Airbnb lodging price dataset in Boston, AutoData can improve the regression model by searching useful records from US lodging price datasets in other cities. The participants can pose queries to find lodging price datasets from multiple sources that can further improve the model performance.

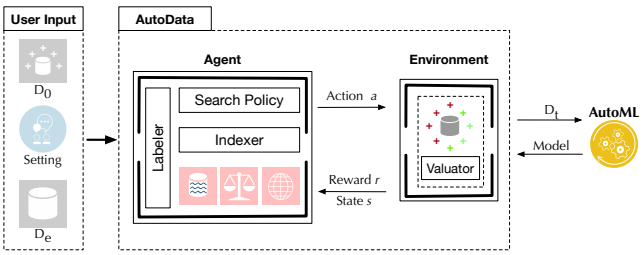

Figure 3: The Architecture of AutoData

2 DEMONSTRATION SCENARIOS

In this section, we introduce two main scenarios — image classification and relational data prediction — for AutoData and demonstrate how AutoData retrieves more fresh data from the repository to improve the model performance.

Settings: The users can specify a task, with datasets description, AutoML framework and goals. **(1) *Data:*** identifies a set of "training" datasets and a representative "test" dataset. Each dataset contains a data source (*e.g.,* a relational table), a data type (*e.g.,* table or image), a boolean value label to indicate whether this dataset is labeled or not. **(2) *AutoML:*** determines the task type (such as Classification or Regression) and the AutoML framework, which is used to build the model(*e.g.,* Auto-Keras, Auto-WEKA). **(3) *Goal:*** sets a termination condition, including the budget for search and performance requirement.

Scenario 1: Image Classification. A user needs to construct a DL model to classify a set of images, which consists of different styles of animal images. She provides 20 images of each class for training and expects a DL model with at least 85% prediction accuracy. In general, AutoData searches images crawled/downloaded from Google, Baidu or ImageNet to improve the model performance, which is low at the beginning because of the small and imbalanced samples. For example, AutoData can supplement images of minor classes or different styles (*e.g.,* cartoon, hand-painted), so as to improve the quality of the training dataset.

(1) Dataset. The user provides 10 classes of animals and 20 sample images for each class as the training set. Also, she makes a test set, in which there are 100 different images for each class. AutoData cannot see the test data, but can test on it and get the performance of model. The data repository of AutoData is multi-source and heterogeneous, from data lakes, data markets and the Web.

(2) Task Specification. The user can provide dataset and specify the settings through the web interface (Figure 2-❶). For example, the user specifies an image classification task and provides a test set together with a training set, aiming to obtain a DL model with an accuracy of over 85%.

(3) Data Search. AutoData first initialize a model on the training set. Next, AutoData will evaluate the influence score of each data point. For example, as shown in Figure 2-❷, we can observe that the training set is unbalanced in image styles, *i.e.,* most of them are images of real animals while other styles (*e.g.,* cartoon) are minor. This situation will be implicitly captured by AutoData because the

images of other styles (*e.g.,* cartoon) will get higher influence scores. Then the agent module of AutoData will pay more attention to these minority. That is AutoData will retrieve more images with other styles (*e.g.,* cartoon) from the data repository. In this way, AutoData can successfully guess the *unknowns* and improve the performance of classification task (Figure 2-❹).

(4) Results. A user can download the model with best performance and data after the system stops. She can check the running status of the system at any time (Figure 2-❸), *e.g.,* the change of the model performance. Once the model satisfies the goal specified by the user or the budget is used up, AutoData terminates this task.

Scenario 2: Relational Data Prediction: A user wants to use the DL model to predict the Airbnb lodging price in the US, which is affected by geographical location. She only obtains the lodging price dataset in Boston. AutoData can derive new data points from other US cities(*e.g.,* LA, NYC, etc) to improve the regression model, which are related to the user input.

For relational data, AutoData first unions related multi-source heterogeneous data together [14] in the data repository. Similar to image data, AutoData valuates the influence of each data point and retrieves new data based on the top-k influential data points. For example, since NYC and Boston have similar prices of lodging, AutoData retrieves some data points there and improves the performance of the model.

3 SYSTEM ARCHITECTURE

3.1 Overview

When a user provides a small training set, AutoData is designed to search for more data to supplement it based on our reinforcement learning framework. AutoData takes as input $(D_0, Setting, D_e)$ and outputs a final model with the best performance by iteratively searching more data. We first briefly introduce the key components of our RL framework and then show an overall pipeline of AutoData.

Environment. In each iteration, environment takes as input training dataset, feeds it into AutoML and outputs the model performance. Besides, it has a *Valuator* that computes influences of all data points on the model. A data point with high influence indicates that changing the point is likely to change the model a lot if we retrain the model.

State(s) denotes the current model performance and influence scores of all data points of the training set.

Reward(r) is the performance change of the model between adjacent iterations, computed by $acc_t - acc_{t-1}$.

Agent. It takes as input the state and the reward, and outputs the retrieved data points that are likely to improve the model performance. This step is conducted by *Search-Policy*, considering the relations between repository data and training data with high influences, as well as the model performance. Besides, the retrieved data may be unlabeled, so *Labeler* can be used to label them [3, 15]. The *Indexer* is utilized to accelerate the search process [11].

Action(a) denotes a set of b data points retrieved from the data repository, which will be added to the training set.

Suppose that D_0 denotes the input dataset and D_e denotes the test set. As shown in Figure 3, the user input is first sent to the **agent**, where the *Labeler* labels the dataset if D_0 is unlabeled. Then D_0 is fed into AutoML in the **environment**, where the subscript denotes the t-th($t = 0$) training iteration. After AutoML builds the model, *Valuator* computes influences of all data points in D_t on the model. These influences as well as the model performance on D_e(**state**) are sent to the **agent**. Then *Search-Policy* takes an **action**, i.e., efficiently fetching a batch of b data points through the index, adds them to form a new training set D_{t+1} and feeds it into AutoML. Agent leverages Q-Table to estimate the long-term **reward**(*i.e.,* expected performance improvement) of each action, and to select the optimal action with the largest long-term reward. The above steps iterate until the terminating condition is achieved.

3.2 Valuator

In each iteration, the performance returned by AutoML can roughly describe current **state** of the environment, but it is not enough to guide the search. We should consider the characteristics of data points in the training set for the ML task. To this end, *Valuator* is proposed to compute the influence score of each data point on current ML model using the influence function [9, 16].

Given a data point x, a high influence score $I(x)$ indicates that the model will change a lot if x has a minor change. This means that x is a "weak point" of the model that is not fitted well, so it should get more attention [9]. Specifically, the model change is described as the parameter change, i.e., $|\theta - \theta'|$, where θ' is the optimized parameter for training data containing x with a minor change, i.e., $x+\epsilon$, and θ is the original parameter. However, retraining the model for each modified x is prohibitively slow. Hence, based on [9], we compute $I(x)$ directly without retraining

$$I(x) = -H_\theta^{-1} \nabla_\theta L(x, \theta) \tag{1}$$

where $H_\theta = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 L(x_i, \theta)$ is the Hessian matrix, n is the size of D_t, $x_i \in D_t$ and L is the loss function. Then we can compute the influence score for each data point and model the **state** in a more fine-grained way, which will be discussed next.

Encoding *state* using *Valuator*. In t-th iteration, after AutoML constructing a new model, the *Valuator* computes the score $I(x), x \in D_t$, which will be normalized to [0,1]. Since the data points with high influence scores have a large impact on the model, we select top-k ones as signals to be sent to the **Agent**. The state can be represented by $s = [(x_1, I(x_1)), ..., (x_k, I(x_k))]$, where $s_i = (x_i, I(x_i))$ and $I(x_i)$ denotes the i-th largest influence score of data x_i.

For efficiency, since $I(x_i)$ is infinite and similar states are likely to generate similar actions, we map influence scores into ten equi-depth sub-ranges of [0,1] so that the search can be more efficient. Here, we abuse $I(x_i)$ to denote the sub-range that x_i lies in. For example, $s_1 = (x_1, [0.9, 1])$ denotes that x_1 is the data point with the largest influence score $I(x_1)$ which lies in [0.9,1]. The state can also be extended by adding the model performance, which will be discussed in Section 3.3.

3.3 Search Policy

Given the **state** as input, the **Agent** uses the *Search-Policy* to choose the optimal **action** with the expected largest reward, i.e., leading to the highest performance improvement.

Constructing the *state*. Recap from Section 3 that the action is defined as a set of b data points retrieved from the repository, but the action space C_N^b is extremely large because the cardinality(N) of the repository is large. Therefore, we simplify this problem by retrieving a data point corresponding to each training point in s independently, and thereby $b = k$. Then the action a is composed of a set of sub-actions, *i.e.,* $a = \{a_1, a_2, \cdots, a_k\}$. At a high level, a_i aims to retrieve a new data point from the repository corresponding to s_i.

Then we need to build the relation between the repository data and training data, so as to guide the search. To this end, we propose to leverage the distance(denoted by $dst(x, x')$) between data points to model the relation. For image data, we can use the cosine similarity between the embedding vectors of two images as the distance. For relational data, Euclidean distance between tuples can be computed as the distance. Similar to the influence score, we normalize the distance to [0,1] and divide it into ten equi-depth sub-ranges. And more specifically, given s_i, the action a_i aims to select a sub-range δ and sample a data point x_i', s.t. $dst(x_i, x_i') \in \delta$.

For example, suppose that the distance is divided into sub-ranges [0,0.1), [0.1,0.2),..., [0.9,1]. Given $s_2 = (x_2, [0.8, 0.9))$ for an image x_2, we compute $a_2 = [0.9, 1]$ using the **policy**. This means that we aim to select an image x_2' in the repository so that $dst(x_2, x_2') \in [0.9, 1]$. The intuition is that these data points with high influence scores serve as the weak points of current model. Therefore, we should search the data similar to this weak point from the repository to make the model more robust.

Next, we introduce how to design the *Search-Policy* to map the state to an optimal action.

***Policy* design.** Then, the *Search-Policy* needs to solve the problem that given x_i, what is the optimal a_i, so that the performance of the final model can be maximized. In order to solve this problem, we consider two aspects: (**A1**) How to estimate the long-term rewards of actions through the learning experience. (**A2**) How to store, learn and update long-term rewards for actions.

To solve **A1**, we use Q-Function $Q(s_i, a_i)$ to estimate the long-term rewards of a_i, which indicates the expected improvement of the model performance by taking a_i under the state s_i, which is computed by

$$Q_t(s_i, a_i) = Q_{t-1}(s_i, a_i) + \alpha\big(r + \gamma \max_{a_i'} Q(s_i', a_i') - Q_{t-1}(s_i, a_i)\big) \quad (2)$$

where t denotes the t-th iteration, α is the learning rate and γ is the discount factor. Then, *Search-Policy* chooses the sub-actions with the largest $Q(s_i, a_i)$ to maximize the model performance.

To solve **A2**, we use Q-Table to store long-term rewards and apply the $\epsilon - greedy$ strategy to learn and update the rewards. To be specific, a 2-dimensional table is utilized to record the map from states to actions. In the table, each row represents the state s_i and each column represents the action a_i. Correspondingly, we record the long-term reward $Q(s_i, a_i)$ in each cell of the table. For example, as shown in Figure 4, given $s_2 = (x_2, [0.9, 1.0])$ and $a_2 = [0.8, 0.9)$, then $Q(s_2, a_2) = 0.05$, that means the model may gain 5% performance improvement when choosing a_2 for s_2.

$I(x)$ \ δ	[0.9,1.0]	[0.8,0.9)	[0.7,0.8)	[0.6,0.7)
[0.9,1.0]	0.02	0.05	0.01	0.01
[0.8,0.9)	0.01	0.01	0	0
[0.7,0.8)	0.03	0.01	0.02	0
[0.6,0.7)	0.01	0	0	0.01

Figure 4: An Example of Q-Table

Then, we use the $\epsilon - greedy$ strategy to balance exploration and exploitation, so that *Search-Policy* can obtain enough experiences to learn the long-term rewards. In the beginning, there are no experiences to make a reasonable decision. Thus, ϵ is set to a larger value(*e.g.,* 0.9). Then, when choosing an action, *Search-Policy* has a 90% probability of randomly choosing an action. Therefore, *Search-Policy* can explore more different actions. However, in the later period, to make the agent learn more from existing experience and use of the learned search strategy, ϵ should decrease.

Besides, when taking model performance in state s into consideration, we can easily extend Q-table to three-dimensional to learn a more fine-grained search policy.

Acknowledgements. This project is supported by NSF of China (61925205, 61632016), Huawei, TAL education, China National Postdoctoral Program for Innovative Talents (BX2021155) , China Postdoctoral Science Foundation(2021M691784), and Zhejiang Lab's International Talent Fund for Young Professionals.

REFERENCES

[1] C. Chai, L. Cao, G. Li, J. Li, Y. Luo, and S. Madden. 2020. Human-in-the-loop Outlier Detection. In *SIGMOD 2020*. ACM, 19–33.
[2] C. Chai, J. Fan, and G. Li. 2018. Incentive-Based Entity Collection Using Crowdsourcing. In *ICDE 2018*. 341–352.
[3] C. Chai, J. Fan, G. Li, J. Wang, and Y. Zheng. 2019. Crowdsourcing Database Systems: Overview and Challenges. In *ICDE 2019*. IEEE, 2052–2055.
[4] C Chai, G Li, J Li, D Deng, and J Feng. 2016. Cost-Effective Crowdsourced Entity Resolution: A Partial-Order Approach. In *SIGMOD 2016*. ACM, 969–984.
[5] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. 2018. A partial-order-based framework for cost-effective crowdsourced entity resolution. *VLDB J.* 27, 6 (2018), 745–770.
[6] F. Chirigati, Rémi Rampin, Aécio S. R. Santos, A. Bessa, and J. Freire. 2021. Auctus: A Dataset Search Engine for Data Augmentation. *CoRR* abs/2102.05716 (2021).
[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR, 2009*.
[8] Z. Gharibshah, X. Zhu, A. Hainline, and M. Conway. 2020. Deep Learning for User Interest and Response Prediction in Online Display Advertising. *DSE* (2020).
[9] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *ICML, 2017*.
[10] M. Li, H. Wang, and J. Li. 2020. Mining Conditional Functional Dependency Rules on Big Data. *Big Data Mining and Analytics* 03, 01, Article 68 (2020), 16 pages.
[11] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Trans. Knowl. Data Eng.* (2020).
[12] Y. Luo, X. Qin, N. Tang, and G. Li. 2018. DeepEye: Towards Automatic Data Visualization. In *ICDE 2018*. 101–112.
[13] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *In SIGMOD 2021*. 1235–1247.
[14] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* (2018).
[15] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proc. VLDB Endow.* (2017).
[16] S. Tian, S. Mo, L. Wang, and Z. Peng. 2020. Deep Reinforcement Learning-Based Approach to Tackle Topic-Aware Influence Maximization. *DSE* 5, 1 (2020), 1–11.