6th ICML Workshop on Automated Machine Learning (2019)

Automatic Machine Learning by Pipeline Synthesis using Model-Based Reinforcement Learning and a Grammar

Iddo Drori IDRORI@NYU.EDU
Yamuna Krishnamurthy * YAMUNA@NYU.EDU
Raoni de Paula Lourenco RAONI@NYU.EDU
Remi Rampin REMI.RAMPIN@NYU.EDU
Kyunghyun Cho KYUNGHYUN.CHO@NYU.EDU
Claudio Silva CSILVA@NYU.EDU
Juliana Freire JULIANA.FREIRE@NYU.EDU
New York University, Tandon School of Engineering and Center for Data Science

Abstract

Automatic machine learning is an important problem in the forefront of machine learning. The strongest AutoML systems are based on neural networks, evolutionary algorithms, and Bayesian optimization. Recently AlphaD3M reached state-of-the-art results with an order of magnitude speedup using reinforcement learning with self-play. In this work we extend AlphaD3M by using a pipeline grammar and a pre-trained model which generalizes from many different datasets and similar tasks. Our results demonstrate improved performance compared with our earlier work and existing methods on AutoML benchmark datasets for classification and regression tasks. In the spirit of reproducible research we make our data, models, and code publicly available.

1. Introduction

Machine learning has three main axes: dataset, task, and solution. Given a dataset, a well-defined machine learning task, and an evaluation criteria, the goal is to solve the task with respect to the dataset while optimizing performance. There are Automatic machine learning (AutoML) (Hutter et al., 2019) problems of increasing difficulty, starting with hyperparameter optimization of a specific algorithm, to the selection of algorithms and their hyperparameter optimization, and finally meta learning, which entails synthesizing an entire machine learning pipeline from machine learning primitives. In this work we present a system that automates model discovery, solving well-defined machine learning tasks on unseen datasets by learning to generate machine learning pipelines from basic primitives.

Our goal is to search within a large space of machine learning primitives and parameters, which together constitute a pipeline, for solving a task on a given dataset. The challenge is that the search space of pipelines, primitives and hyperparamaters is high dimensional. Similarly, programs for playing board games such as chess and Go are faced with the challenge of searching in a high dimensional space. We formulate the AutoML problem of pipeline synthesis as a single-player game, in which the player starts from an empty pipeline, and in each step is allowed to perform edit operations to add, remove, or replace pipeline components according to a pipeline grammar. A sequence of game steps results in a complete pipeline which is executed on the dataset to solve the task, evaluated by pipeline performance. Formally, an entire pipeline is a state, an action corresponds to modifying the

∗. Y. Krishnamurthy is currently a doctoral candidate at Royal Holloway, University of London.

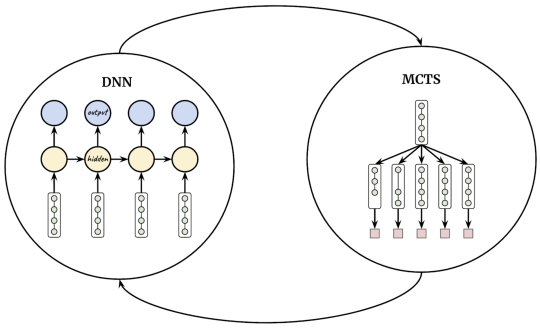


Figure 1: Architecture: the neural network sequence model (left) receives an entire pipeline, meta features, and task as input. The network estimates action probabilities and pipeline evaluations. The MCTS (right) uses the network estimates to guide simulations which terminate at actual pipeline evaluations (square leaf nodes).

current pipeline to derive a new pipeline, and pipeline performance is the reward. Thus, our approach is based on reinforcement learning (Sutton and Barto, 2018).

An inherent advantage of this approach is that we have the provenance of all actions and decisions leading to a given pipeline, which provides an explanation for the synthesis process. Early programs for chess and Go searched a large space of millions of positions. AlphaZero (Silver et al., 2018) reduced this search by three orders of magnitude using a general reinforcement learning algorithm. Inspired by expert iteration (Anthony et al., 2017) we use a neural network for predicting pipeline performance and action probabilities along with a Monte-Carlo tree search (MCTS) which makes strong decisions based on the network, as shown in Figure 1. The process progresses by self play with iterative self improvement, and is known to be highly efficient at finding a solution to search problems in high dimensional spaces. Considering all combinations of machine learning primitives would result in many invalid pipelines. Therefore, we use a pipeline grammar which reduces the search space even further. Using a grammar decreases the branching factor and average depth of the MCTS. Our approach allows to learn to synthesize a pipeline from scratch, tabula rasa, or to generalize from many different datasets and similar tasks by using a pre-trained neural network. The main contributions of this work are:

1. **Pipeline grammar**: reducing the branching factor and average search depth. We compare performance across time, with and without a grammar.

2. **Pre-trained model**: generalizing from many different datasets and similar tasks in contrast with learning from scratch every time.

3. **Meta learning minute**: we investigate performance using benchmark AutoML datasets across time, focusing on the first minutes of computation. Our results demonstrate competitive performance with computation time which is an order of magnitude faster than the latest AutoSklearn.

4. **Open source**: in the spirit of reproducible research we make our data, models, and code publicly available (Drori et al., 2019).

Existing AutoML systems use differentiable programming (Milutinovic et al., 2017), tree search (Swearingen et al., 2017), evolutionary algorithm (Olson and Moore, 2016; Chen et al.,

2018; Gijsbers et al., 2017), Bayesian optimization (Rasmussen, 2003; Bergstra and Bengio, 2012; Feurer et al., 2015; Kotthoff et al., 2017), collaborative filtering (Yang et al., 2018; Fusi et al., 2018), and grammars (de Sá et al., 2017), for finding machine learning pipelines for a given dataset and task. Our system uses model-based reinforcement learning, combining a neural network with MCTS, and the search is constrained by a pipeline grammar.

2. Methods

Our earlier work on AlphaD3M (Drori et al., 2018), formulates the AutoML problem of meta learning as pipeline synthesis using a single-player game with a neural network sequence model and Monte Carlo tree search (MCTS). A pipeline is a data mining work flow, of data pre-processing, feature extraction, feature selection, estimation, and post-processing primitives. Possible states are all valid pipelines generated from a set of primitives, constrained on actions of insertion, deletion or substitution of a primitive in the pipeline. To reduce the search space, we define a pipeline grammar where the rules of the grammar constitute the actions. The grammar rules grow linearly with the number of primitives and hence address the issue of scalability. Our architecture models meta data and an entire pipeline chain as state rather than individual primitives. A pipeline, together with the meta data and problem definition is analogous to an entire game board configuration. The actions are transitions from one state (pipeline) to another, defined by the production rules of the grammar.

2.1 Neural Network and Monte-Carlo Tree Search

Our system uses a recurrent neural network shown in Figure 1 (left), specifically a long short-term memory (LSTM) sequence model (Hochreiter and Schmidhuber, 1997), defined as $(p, v) = f_\theta(s)$ with parameters θ. The network $f_\theta(s)$ receives a pipeline state representation as input s and computes a vector of probabilities $p_\theta = P(a|s)$ over all valid actions a, and a value $v = \mathbb{E}[e|s]$ which approximates the pipeline evaluation e when run on the data for solving the task. The system learns these action probabilities and the estimated values from games of self-play which guide the search in future games. The parameters θ are updated by stochastic gradient descent on the following loss function:

$$L(\theta) = -\pi \log p + (v - e)^2 + \alpha \|\theta\|^2, \tag{1}$$

maximizing cross entropy between policy vector p and search probabilities π, minimizing mean squared error between predicted performance v and actual pipeline evaluation e, and regularizing the network parameters θ to avoid over fitting. Our system uses Monte-Carlo tree search which is a stochastic search using upper confidence bound update rule:

$$U(s, a) = Q(s, a) + cP(a|s)\frac{\sqrt{N(s)}}{1 + N(s, a)}, \tag{2}$$

where $Q(s, a)$ is the expected reward for action a from state s, $N(s, a)$ is the number of times action a was taken from state s, $P(a|s)$ is the estimate of the neural network for the probability of taking action a from state s, and c is a constant which determines the amount of exploration. At each step of the simulation, we find the action a and state s which maximize $U(s, a)$ and add the new state to the tree, if it does not exist, with the neural network estimates $P(a|s), v(s)$ or call the search recursively. Finally, the search terminates

Table 1: Grammar $< T, N, P, S >$ for machine learning pipelines for a classification task.

T[Terminals]	$SkImputer$, $MissingIndicator$, $OneHotEncoder$, $OrdinalEncoder$, PCA ..., $GaussianNB$, $RidgeClassifier$, $SGDClassifier$, $LinearSVC$
N[Non-Terminals]	$DataCleaning$ <DC>, $DataTransformation$ <DT>, $Estimators$ <E>
S[Start]	S
P[Production Rules]	<S> ::= <E> | <DC> <E> | <DT> <E> | <DC> <DT> <E> <DC> ::= $SkImputer$ <DC> |...| $MissingIndicator$ <DC> | $SkImputer$ |...| $MissingIndicator$ <DT> ::= $OneHotEncoder$ <DT> $OrdinalEncoder$ <DT> |...| PCA <DT> | $OneHotEncoder$ $OrdinalEncoder$ |...| PCA <E> ::= $GaussianNB$ | $RidgeClassifier$ | $SGDClassifier$ |...| $LinearSVC$

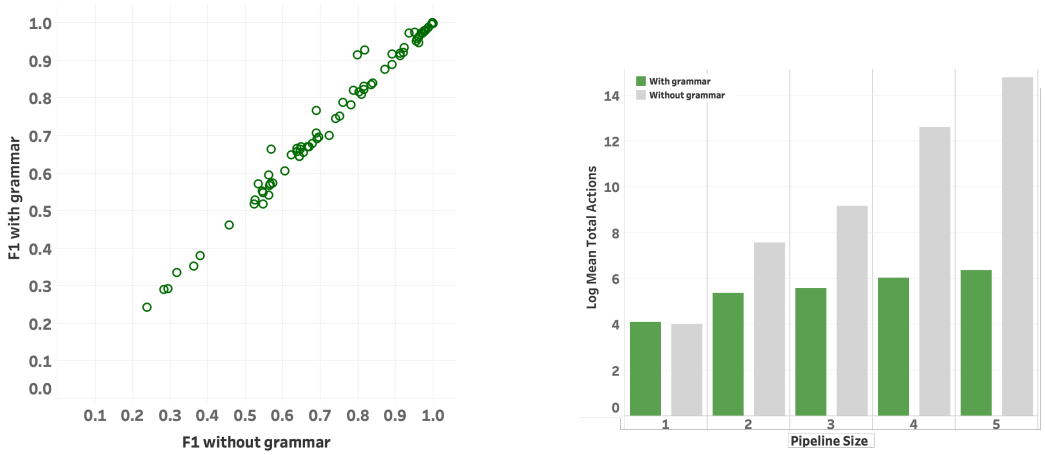
and a pipeline is realized and applied to the data to solve the task, resulting in an actual evaluation e of running the generated pipeline on the data and task.

2.2 Pipeline Grammar

We use a context free grammar (CFG) (Chomsky, 1956) to add domain knowledge for generating valid pipelines represented as strings. A CFG is represented by a four-tuple $< T, N, P, S >$ where T is the set of terminals which are the components that make the string, N the set of non-terminals which are placeholders for patterns of terminals that are generated from them, P the set of production rules for replacing a non-terminal with other non-terminals or terminals and S the start symbol, which is a special non-terminal symbol that appears in the initial string generated by the grammar. We use the prior knowledge of working pipelines for specific tasks in constructing the CFG for generating machine learning pipelines for that task. For example, for classification and regression tasks the pipeline usually consists of data cleaning, data transformation and an estimator (classifier or regressor) primitives in that order. Data cleaning consists of primitives that fix the data to make it suitable as inputs to the estimators, such as imputing missing values. Data transformation primitives transform the original data, for example, dimensionality reduction, categorical value encoding, and feature selection. Estimators are the learning primitives, which for a classification task are a set of classifiers such as Naive Bayes and SVM, and for regression are a set of regressors such as Linear and Ridge Regression. Based on this sequence of components for classification and regression tasks we define the CFG, using Sklearn (Pedregosa et al., 2011) primitives, for classification as shown in Table 1. Our system is configurable with different grammars for a given task.

3. Results

For evaluation we used 296 tabular datasets from OpenML (Vanschoren et al., 2014) consisting of both classification and regression tasks. For our comparison with AutoSklearn as a baseline (Guyon et al., 2019), we used only comparable Sklearn primitives, specifically 2 data cleaning, 11 data transformation, 16 classification and 22 regression primitives. While AutoSklearn is limited to using the Sklearn library, AlphaD3M can be configured with machine learning primitives from other libraries.



(a) (b)

Figure 2: (a) Comparison of performance with and without using a pipeline grammar: Each point represents an OpenML dataset. Performance is not degraded even though computation time is reduced. (b) Comparison of log mean total actions with and without a grammar.

3.1 Pipeline Grammar

Figure 2(a) compares performance with and without a pipeline grammar. Each point represents one of 74 different OpenML datasets on a classification task, showing that performance is maintained, across the diagonal, even though using a grammar significantly reduces the search space. Figure 2(b) compares the logarithm of the mean of total actions with and without a pipeline grammar, showing a significant reduction in the search space size. Using the pipeline grammar is on average between two and three times as fast than without using the grammar. Using the grammar reduces the average depth of the MCTS by an order of magnitude and decreases the branching factor of the MCTS on average by three-fold compared to the non-grammar.

3.2 Pre-trained Model

Comparing between AutoML methods requires taking into account both performance and running time. Given sufficient running time, most methods search the space until reaching an optimal solution, so our interest is in comparing between the efficiency of different methods. We therefore compare performance across multiple running times, which unveils how the methods progress. Specifically the most interesting difference occurs during the first minutes of computation. Figure 3 compares performance between AlphaD3M using a grammar and a model pre-trained on other datasets (dark green), AlphaD3M using a grammar trained from scratch tabula rasa (light green), and AutoSklearn (gray). We used the same Sklearn machine learning primitives for both AlphaD3M and AutoSklearn, on a sample of benchmark AutoML datasets, running on a cluster with 4 Tesla 100 GPU's. The horizontal axis denotes time in 2^i seconds for $i = 1, \ldots, 8$, (ie left is better by an exponential factor in time). The vertical axis denotes F1-score in $[0, 1]$, (ie higher is better). Our pre-trained model meta learns from other datasets and quickly generalizes reaching a result within 4 seconds (blue), including meta feature computation time. This is twice as fast as learning from scratch

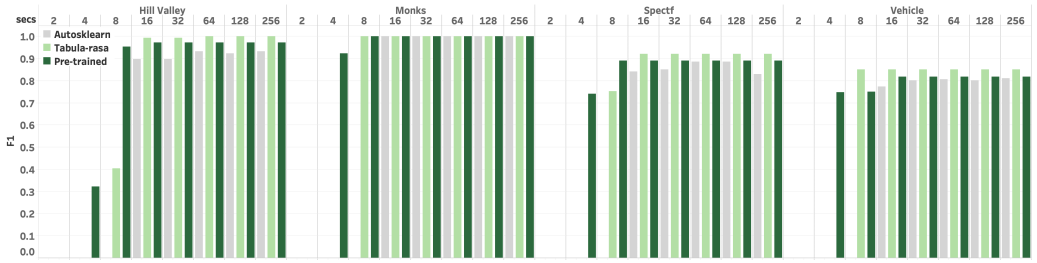


Figure 3: Performance-time comparison between (i) AlphaD3M using a pre-trained model and a grammar, (ii) model trained tabula rasa and a grammar, (iii) AutoSklearn. All methods (including brute force) perform comparably given sufficient time using same primitives, the difference is in performance given equal times. Method (i) is faster than (ii) which in turn is faster than (iii). Performance is F1 and time is in seconds on an exponential scale.

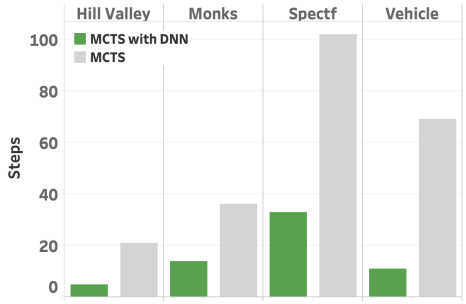


Figure 4: Ablation comparison between number of steps of MCTS with NN vs. MCTS only.

tabula rasa and our results demonstrate improvement in performance compared to learning the network from scratch. In turn, our method learning tabula rasa reaches a result twice as fast as AutoSklearn, with comparable performance. In AlphaD3M and AlphaZero, the neural network policy converges in the limit to the Monte-Carlo tree search (MCTS) policy, however the difference is in efficiency. Figure 4 compares the number of steps required for reaching the same performance using only MCTS with MCTS and a neural network.

4. Conclusions

We extended AlphaD3M, an automatic machine learning system by using a pipeline grammar and a pre-trained model. We analyzed the contribution of our extensions, comparing performance with and without a grammar, learning from scratch and using a pre-trained model, across time at fine granularity. Our results demonstrate competitive performance while being an order of magnitude faster than existing state-of-the-art AutoML methods. Our system is open, supporting any set of machine learning primitives, such as D3M, Sklearn, or general operations, thus opening the door for other application domains. In the spirit of reproducible research we make the data, models, and code available (Drori et al., 2019).

Acknowledgements

This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) Data-Driven Discovery of Models (D3M) Program.

References

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pages 5360–5370, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, pages 281–305, 2012.

Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: A compositional evolutionary learning system. *The Genetic and Evolutionary Computation Conference*, pages 402–409, 2018.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Alex GC de Sá, Walter José GS Pinto, Luiz Otavio VB Oliveira, and Gisele L Pappa. Recipe: a grammar-based framework for automatically evolving classification pipelines. In *European Conference on Genetic Programming*, pages 246–261. Springer, 2017.

Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. AlphaD3M: Machine learning pipeline synthesis. *ICML International Workshop on Automatic Machine Learning*, 2018.

Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Code for automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar, 2019. URL `https://www.dropbox.com/sh/zgvr6revq6qyb5w/AAAJc5IMYOVHaM3di3cxhjMaa?dl=0`.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2755–2763, 2015.

Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*, pages 3348–3357, 2018.

Pieter Gijsbers, Joaquin Vanschoren, and Randal S Olson. Layered TPOT: Speeding up tree-based pipeline optimization. *Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms*, 2017.

Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michéle Sebag, Alexander Statnikov, WeiWei Tu, and Evelyne Viegas. Analysis of the automl challenge series 2015-2018. In *AutoML*, Springer series on Challenges in Machine Learning, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(1):826–830, 2017.

Mitar Milutinovic, Atlm Gunes Baydi, Robert Zinkov, William Harvey, Dawn Song, and Frank Wood. End-to-end training of differentiable pipelines across machine learning frameworks. In *NeurIPS Autodiff Workshop*, 2017.

Randal Olson and Jason Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on Automatic Machine Learning*, pages 66–74, 2016.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct): 2825–2830, 2011.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, Shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Thomas Swearingen, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross, and Kalyan Veeramachaneni. ATM: A distributed, collaborative, scalable system for automated machine learning. In *IEEE International Conference on Big Data*, pages 151–162, 2017.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. OBOE: Collaborative filtering for AutoML initialization. *NeurIPS Workshop on Meta-Learning*, 2018.