Supplementary Material for MOAZ: A Multi-Objective AutoML-Zero Framework

Ritam Guha
guharita@msu.edu
Michigan State University
East Lansing, Michigan, USA

Wei Ao
aowei@msu.edu
Michigan State University
East Lansing, Michigan, USA

Stephen Kelly
spkelly@mcmaster.ca
McMaster University
Hamilton, Ontario, Cananda

Vishnu Boddeti
vishnu@msu.edu
Michigan State University
East Lansing, Michigan, USA

Erik Goodman
goodman@msu.edu
Michigan State University
East Lansing, Michigan, USA

Wolfgang Banzhaf
banzhafw@msu.edu
Michigan State University
East Lansing, Michigan, USA

Kalyanmoy Deb
kdeb@egr.msu.edu
Michigan State University
East Lansing, Michigan, USA

ACM Reference Format:
Ritam Guha, Wei Ao, Stephen Kelly, Vishnu Boddeti, Erik Goodman, Wolfgang Banzhaf, and Kalyanmoy Deb. 2023. Supplementary Material for MOAZ: A Multi-Objective AutoML-Zero Framework. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3583131.3590391

1 FLOP COUNTS

We have approximated the Floating Point Operations (FLOP) count for each of 65 operations used in AutoML-Zero. These counts are listed in Table 1.

2 ADDITIONAL DESCRIPTION OF NON-LINEAR PROBLEM

In addition to the results mentioned in the main manuscript, here we provide some additional results that we obtained for the non-linear problem. The non-linear problem is formulated according to the process discussed in [1]. A teacher neural network acts as a non-linear regression model represented as $L(x_i) = u.ReLU(Mx_i)$ where M is a random 8×8 matrix and u is a random vector. In AZ/MOAZ nomenclature, this teacher neural network can be represented as shown in Figure 1. The task of AZ/MOAZ is to rediscover this neural network by getting signals from the dataset created by using the teacher neural network.

Table 1: The approximated number of FLOP for 65 operations used in AutoML-Zero. $VSize$ and $MSize$ refer to the dimensions of vectors and matrices used in the operations, respectively.

Operators	Approx. FLOP	Operators	Approx. FLOP
NO_OP	0	MATRIX_MAX_OP	$MSize$
SCALAR_SUM_OP	1	MATRIX_ABS_OP	$MSize$
SCALAR_DIFF_OP	1	MATRIX_HEAVYSIDE_OP	$MSize$
SCALAR_PRODUCT_OP	1	MATRIX_CONST_SET_OP	$MSize$
SCALAR_DIVISION_OP	1	SCALAR_VECTOR_PRODUCT_OP	$VSize$
SCALAR_MIN_OP	1	VECTOR_INNER_PRODUCT_OP	$2*VSize$
SCALAR_MAX_OP	1	VECTOR_OUTER_PRODUCT_OP	$VSize*VSize$
SCALAR_ABS_OP	1	SCALAR_MATRIX_PRODUCT_OP	$MSize*MSize$
SCALAR_HEAVYSIDE_OP	1	MATRIX_VECTOR_PRODUCT_OP	$MSize*MSize$
SCALAR_CONST_SET_OP	1	VECTOR_NORM_OP	$2*VSize$
SCALAR_SIN_OP	1	MATRIX_NORM_OP	$MSize*MSize$
SCALAR_COS_OP	1	MATRIX_TRANSPOSE_OP	$MSize$
SCALAR_TAN_OP	1	MATRIX_MATRIX_PRODUCT_OP	$MSize*MSize$
SCALAR_ARCSIN_OP	1	VECTOR_MEAN_OP	$MSize*MSize$
SCALAR_ARCCOS_OP	1	VECTOR_ST_DEV_OP	$MSize*MSize$
SCALAR_ARCTAN_OP	1	MATRIX_MEAN_OP	$MSize*MSize$
SCALAR_EXP_OP	1	MATRIX_ST_DEV_OP	$MSize*MSize$
SCALAR_LOG_OP	1	MATRIX_ROW_MEAN_OP	$MSize$
VECTOR_SUM_OP	$VSize$	MATRIX_ROW_ST_DEV_OP	$MSize$
VECTOR_DIFF_OP	$VSize$	SCALAR_GAUSSIAN_SET_OP	1
VECTOR_PRODUCT_OP	$2*VSize$	VECTOR_GAUSSIAN_SET_OP	$VSize$
VECTOR_DIVISION_OP	$VSize$	MATRIX_GAUSSIAN_SET_OP	$MSize*MSize$
VECTOR_MIN_OP	$VSize$	SCALAR_UNIFORM_SET_OP	1
VECTOR_MAX_OP	$VSize$	VECTOR_UNIFORM_SET_OP	$VSize$
VECTOR_ABS_OP	$VSize$	MATRIX_UNIFORM_SET_OP	$MSize*MSize$
VECTOR_HEAVYSIDE_OP	$VSize$	SCALAR_RECIPROCAL_OP	1
VECTOR_CONST_SET_OP	$VSize$	SCALAR_BROADCAST_OP	1
MATRIX_SUM_OP	$MSize$	VECTOR_RECIPROCAL_OP	$VSize$
MATRIX_DIFF_OP	$MSize$	MATRIX_RECIPROCAL_OP	$MSize*MSize$
MATRIX_PRODUCT_OP	$MSize*MSize$	MATRIX_ROW_NORM_OP	$MSize$
MATRIX_DIVISION_OP	$MSize$	MATRIX_COLUMN_NORM_OP	$MSize$
MATRIX_MIN_OP	$MSize$	VECTOR_COLUMN_BROADCAST_OP	$VSize$
VECTOR_ROW_BROADCAST_OP	$VSize$		

2.1 Experimental Setting

To perform the experiments with MOAZ, we used the following lower and upper bounds on complexity and error: the lower bound was 0 for both objectives, whereas the upper bound was 200 for complexity and 0.4 for error. We permit all instructions used by the teacher neural network in the search space. A target error for the problem (e_T) is defined as 5×10^{-2}. So, a run is considered to be successful if it could find at least one algorithm that has less than 5×10^{-2} error. Both AZ and MOAZ are run 30 times with

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07…$15.00
https://doi.org/10.1145/3583131.3590391

```
# sX/vX/mX: scalar/vector/matrix memory
# at address X.
def Setup():
 s2 = 0
def Predict(v0):
 v2 = dot(m0, v0)
 v3 = maximum(v2, v4)
 s1 = dot(v3, v1)
def Learn(v0, s0):
 s3 = s0 - s1
 s3 = s2 * s3
 v5 = s3 * v3
 v1 = v1 + v5
 v6 = s3 * v1
 v7 = heaviside(v2, 1.0)
 v6 = v7 * v6
 m1 = outer(v6, v0)
 m0 = m0 + m1
```

Figure 1: Illustration of the teacher neural network used as a non-linear regression model for generating labels for non-linear regression.

different random seeds. The results and some comparisons of the performance of both frameworks are provided in Section 2.2.

2.2 Performance Comparison

The combined results for AZ and MOAZ are provided in Figure 2 and Figure 3, respectively. After combining the results for all 30 runs, the non-dominated solutions from all discovered algorithms are identified. These solutions are marked in different colors in the figures.

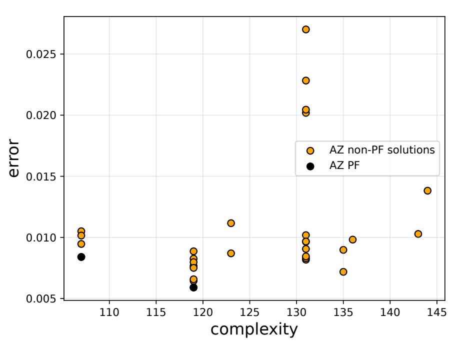

Figure 2: Combined results of AZ runs. Here PF refers to the Pareto Front of the combined results of AZ.

The distribution of errors and complexity is depicted in Figures Figure 4 and Figure 5, respectively. As the error range is $[0, 0.4]$, MOAZ has found a good distribution of solutions in the specified range. In contrast, most of the AZ solutions are concentrated toward the 0-error region. Similarly, for the complexity of solutions, MOAZ has found complexities in the range of $[50, 140]$ whereas all the

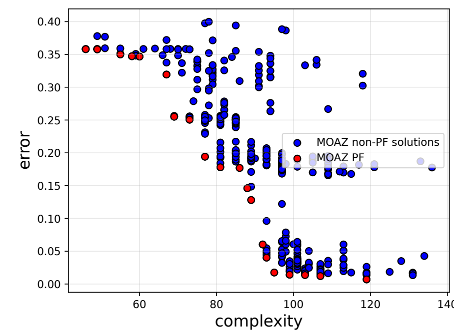

Figure 3: Combined results of MOAZ runs. Here PF refers to the Pareto Front of the combined results of AZ.

solutions are beyond 100 complexity in the case of AZ. One of the interesting applications of finding a broad distribution is platform-based designs. Depending on different platforms, users might want to use different solutions. If the computational capability of a platform is on the lower side, the users can use a low-complexity model with some trade-off in accuracy. This is not possible with AZ solutions because AZ does not take into account complexity. The solutions of MOAZ are statistically significant compared to the AZ solutions both in terms of error and complexity with p-value in the order of 10^{-9}.

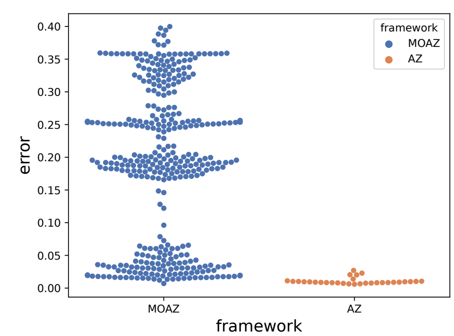

Figure 4: Swarm Plot comparison between AZ and MOAZ for error. In this plot, if the algorithms have equal errors, they are placed on the same line side-by-side.

2.3 Discovered Algorithms

It is not possible to show all the algorithms discovered by both AZ and MOAZ. In this subsection, we are showing one representative algorithm for each of the frameworks. The AZ solution is shown in Figure 6 and the MOAZ solution is presented in Figure 7. There are not many differences between these representative algorithms, apart from the initialization in the *Setup* component. MOAZ solution initializes just the two layers in the neural network but AZ solution initializes some other vectors as well. Please note that these are discovered algorithms with more than 99% accuracy. Some

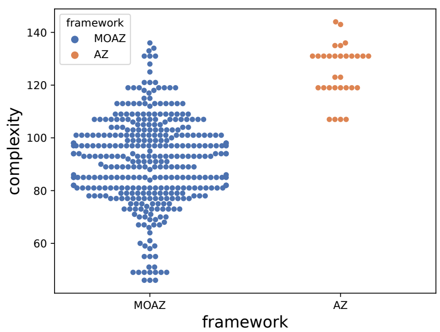

Figure 5: Swarm Plot comparison between AZ and MOAZ for complexity. In this plot, if the algorithms have equal complexity, they are placed on the same line side-by-side.

of these algorithms are very close to the teacher neural network. MOAZ also provides other solutions which are not this close to the teacher neural network but trade off some accuracy for the complexity.

```
# sX/vX/mX: scalar/vector/matrix memory
# at address X.
def Setup():
 v8 = gaussian(2.24632, 0.0306014, n_features)
 m0 = gaussian(0.0295614, 0.0271724, (n_features,
n_features)) # 1st layer weights
 s3 = 0.0350772 # Learning rate
 v4 = gaussian(0.785895, 0.783264, n_features)
 v2 = gaussian(-0.39658, 0.93701, n_features) # 2nd layer
weights
def Predict(v0): # v0: features
 v3 = dot(m0, v0) # Apply 1st layer weights
 v4 = maximum(v3, v6) # Apply ReLU
 s1 = dot(v4, v2) # Apply 2nd layer weights
def Learn(v0, s0): # s0: labels, s1: predictions
 s4 = s0 - s1 # Compute error
 s4 = s3 * s4 # Scale by learning rate
 v6 = s4 * v8
 v2 = v2 + v6 # Update 2nd layer weights.
 v7 = s4 * v2
 v8 = heaviside(v3, 1.0) # ReLU gradient
 v7 = v8 * v7
 m1 = outer(v7, v0)
 m0 = m0 + m1 # Update 1st layer weights.
```

Figure 6: Illustration of a working algorithm for non-linear regression discovered by AZ with a complexity 107.

REFERENCES

[1] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020.

```
# sX/vX/mX: scalar/vector/matrix memory
# at address X.
def Setup():
 v2 = gaussian(-0.0635549, 0.51511, n_features) # 2nd
layer weights
 m0 = gaussian(0.0391599, 0.0467162, (n_features,
n_features)) # 1st layer weights
 s3 = 0.0237734 # Learning rate
def Predict(v0): # v0: features
 v3 = dot(m0, v0) # Apply 1st layer weights
 v4 = maximum(v3, v5) # Apply ReLU
 s1 = dot(v4, v2) # Apply 2nd layer weights
def Learn(v0, s0): # s0: labels, s1: predictions
 s4 = s0 - s1 # Compute error
 s4 = s3 * s4 # Scale by learning rate
 v6 = s4 * v4
 v2 = v2 + v6 # Update 2nd layer weights.
 v7 = s4 * v2
 v8 = heaviside(v3, 1.0) # ReLU gradient
 v7 = v8 * v7
 m1 = outer(v7, v0)
 m0 = m0 + m1 # Update 1st layer weights.
```

Figure 7: Illustration of a working algorithm for non-linear regression discovered by MOAZ with a complexity of 95.