新人ゼミ AutoML-Zero

三嶋隆史

2024 年 05 月 28 日

目次

第 1 章

はじめに

1.1 研究の背景

AutoML は, 機械学習のモデルを自動で最適化する手法として注目されてきた. AutoML 登場以前は, 機械学習のモデルを最適化を人間の手でしていたため, 高度な専門知識と膨大な時間を必要としていた. AutoML はこの膨大にかかる最適化プロセスをコンピュータに置き換えることを目指してきた分野であり [5][7][6], 今後の機械学習の進歩において非常に重要である.

AutoML の試みは非常に有用であったが, これまでの AutoML に関する研究の多くは, 人間のデザインに大きく依存した制約付きの空間上での探索をするものである. 例えば, ニューラルネットワークの構造探索では, 事前に専門家が用意した経験則的に性能が高くなる層を構成要素として使うことで, 最適化の対象を構成要素の組合わせやハイパーパラメータに限定したり, 重みの更新方法は探索せず誤差逆伝搬法を常に用いることで探索空間を制限している [15][10][14]. 他の AutoML でも同様に, 誤差逆伝搬法の学習ルール, データの増強, 強化学習における好奇心に基づく内部報酬といったアルゴリズムの特定の要素のみを最適化対象とすることで探索空間を限定している [2][3][1].

人間のデザインに大きく依存した制約付きの空間上で探索をするアルゴリズムは, 計算コストを抑えられる一方で, 主に 2 つの問題点が存在する. 1 つ目は, 人間がデザインした探索空間にはバイアスがかかってしまい, イノベーションの可能性, すなわち人間がまだ発見していないより良いアルゴリズムを見つけられる可能性が減少してしまう点である. イノベーションは, 探索空間が制限されることにも起因して発生する [4]. 実際, 探索空間の制限は性能に影響を大きく影響を与える観点が無視されることがある [8]. 2 つ目は探索空間を限定する際は極めて慎重に行う必要があり [16][13][9], 結果的に研究者に負担が掛かってしまい, 本来の AutoML の目的でが達成されなくなる点である.

これらの AutoML における問題点を解決するために, 人間からの入力を最小限で機械学習アルゴリズムの探索を行う分野である AutoML-Zero が出現した [11]. Esteban らが提案した AutoML-Zero は, Regularized Evolution (RE) を世代交代モデルを導入した進化計算によって, 機械学習アルゴリズムを探索する手法である. Esteban らの AutoML-Zero では, 機械学習のアルゴリズムを仮想メモリ上で動作するプログラムとして表現する. プログラムは Setup, Predict, Learn の 3 つの関数で構成され, 各関数内の命令では高校数学程度の演算が行われる. RE では突然変異によってアルゴリズムの改善を目指す. 突然変異には, 命令等をランダムに追加または削除したり, 命令に入出力する変数を書き換えたり, Setup, Predict, Learn のいずれかをすべてランダムに初期化したりするオペレータが含まれている. Esteban らの AutoML-Zero は, 人間による事前知識はほとんど使用していないにも関わらず, 勾配降下法や ReLU 関数の再発明に成功している [11].

1.2 研究の目的

Esteban らの AutoML-Zero は人間の入力を最小限にして, 機械学習アルゴリズムを探索するという点については成功であったものの探索効率については多くの問題が残されている．特に我々は, 3 つの問題に着目した．1 つめは世代交代モデルとして RE を使うことによる集団の多様性の低下である．2 つめはアルゴリズムに対する制約が少なすぎることに起因する探索空間の冗長性である．AutoML-Zero では, 人間の入力を最小限にすることは非常に重要であるが, どのような機械学習タスクに対しても性能が良くないアルゴリズムを探索する必要はない．Esteban らの AutoML-Zero は, 突然変異や初期集団の生成を行う際のランダム性が非常に強い上に制約が少ない．そのため, 例えば予測の結果に入力ベクトルを使用しないアルゴリズムまでもが探索対象となっている．3 つ目は, ランダム性が非常に高い突然変異手法である．Esteban らの AutoML-Zero では, アルゴリズムを突然変異させる際に, 実数の定数や命令に入出力する変数を完全にランダムに変更している．ランダム性が高い突然変異には, 集団の評価値の改善を停滞させる作用があると考えられる．

本論文の目的は, Esteban らの AutoML-Zero における集団の多様性維持に関する問題, 探索空間の冗長性に関する問題および突然変異に関する問題に対処した手法を提案し, 線形回帰タスク, 非線形回帰タスク, 分類タスクなどの主要な機械学習タスクにおいて探索が効率的になることを確認することである．

第 2 章

問題の所在

2.1 はじめに

本研究で対象とする問題である AutoML-Zero は, 与えられた機械学習のタスク集合 $\mathcal{T}$ 内のタスクを汎用的に性能高く解くことができるアルゴリズム $a^* \in \mathcal{A}$ を $\mathcal{T}_{\text{search}}$ から探索する問題である. ここで, $\mathcal{A}$ はアルゴリズム全体の集合であり, $\mathcal{T}_{\text{search}}$ は $\mathcal{T}$ に含まれるタスクを偏りなく集めた有限部分集合である. 以下, 第 2.2 節ではタスク集合の厳密な定義, 第 2.3 節では汎用的に性能が高いアルゴリズムの定式化と説明, 第 2.4 節では AutoML-Zero の既存手法の説明を行う.

2.2 タスク集合の定義

タスク集合 $\mathcal{T}$ は複数の機械学習タスクによって構成される. 各機械学習タスク $T^{(i)} \in \mathcal{T}$ は, 入力ベクトル $\boldsymbol{x}_j^{(i)}$ と正解ラベル $y_j^{(i)}$ の順序対の集合であり, 以下のように定義される.

$$T^{(i)} = \left\{ (\boldsymbol{x}_j^{(i)}, y_j^{(i)}) \mid \boldsymbol{x}_j^{(i)} \in \mathbb{R}^{d^{(i)}},\ y_j^{(i)} \in \mathbb{R},\ j \in \mathbb{N},\ 1 \leq j \leq N^{(i)} \right\}$$

ここで, $N^{(i)}$ および $d^{(i)}$ はそれぞれタスクごとに定まるデータの個数とタスクの次元である. また, タスク $T^{(i)}$ には学習用データ $D_{\text{train}}^{(i)} \subset T^{(i)}$ および検証用データ $D_{\text{valid}}^{(i)} \subset T^{(i)}$ が定められており, $D_{\text{train}}^{(i)} \cup D_{\text{valid}}^{(i)} = T^{(i)}$, $D_{\text{train}}^{(i)} \cap D_{\text{valid}}^{(i)} = \emptyset$ を満たす.

2.3 汎用的に性能が高いアルゴリズム

汎用的に性能が高いアルゴリズム $a^* \in \mathcal{A}$ は以下のように定式化される.

$$a^* = \min_{a \in \mathcal{A}} \sum_{T^{(i)} \in \mathcal{T}} l(a, T^{(i)}) \simeq \min_{a \in \mathcal{A}} \sum_{T^{(i)} \in \mathcal{T}_{\text{eval}}} l(a, T^{(i)})$$

ここで, $l(a, T^{(i)})$ はアルゴリズム a を用いて $T_{\text{train}}^{(i)}$ で学習を行った上で, $T_{\text{valid}}^{(i)}$ に対する損失を計算したものである. 損失の算出方法はユーザが設定する. 例えば, 線型回帰のタスクであればアルゴリズム a を使った予測ラベルと正解ラベルの平均二乗誤差, 分類問題であれば分類の正答率等が使われる. 一般に, $\mathcal{T}$ に含まれるタスクをすべて得ることは困難であるため[*1], 損失の総和は $\mathcal{T}$ の有限部分集合 $\mathcal{T}_{\text{eval}}$ を使うことで近似する. 近似の精度が悪化しないようにするために, $\mathcal{T}_{\text{eval}}$ に含まれるタスクも $\mathcal{T}_{\text{search}}$ と同様にタスクの種類が偏らないように注意する必要がある. また,

[*1] 例えば $\mathcal{T}$ が線型回帰タスクの全体である場合, $\mathcal{T}$ の要素数は有限とはならないので, すべて集めることは出来ない.

$\mathcal{T}_{\text{search}}$ にオーバーフィッティングしたアルゴリズムが a^* として選ばれないようにするために, $\mathcal{T}_{\text{search}}$ と $\mathcal{T}_{\text{eval}}$ は独立に $\mathcal{T}$ からタスクを集める.

汎用的に性能が高いアルゴリズムについて具体例を挙げて説明する. $\mathcal{T}$ が線形回帰タスク全体 $\mathcal{T}_{\text{LinReg}}$ である時は, 探索によって線型回帰アルゴリズム $a_{\text{LinReg}} \in \mathcal{A}$ が得られれば, $\mathcal{T}_{\text{LinReg}}$ で内のすべてのタスクに対して誤差を小さく回帰できるため, 汎用的に性能が高いアルゴリズムとなる. しかし, $\mathcal{T}$ が一般の回帰問題全体の集合 $\mathcal{T}_{\text{Reg}}$ である場合は, $\mathcal{T}_{\text{Reg}}$ の中には線型回帰タスクに加えて非線形回帰タスクも含まれている. 故に, 線型回帰アルゴリズム a_{LinReg} が得られたとしても, 非線形回帰問題に対して十分な回帰ができないため, a_{LinReg} は汎用的に性能が高いアルゴリズムにはならない. したがって, 本問題設定ではタスク集合として $\mathcal{T}_{\text{LinReg}}$ を与えたときは線型回帰アルゴリズム $a_{\text{LinReg}} \in \mathcal{A}$, $\mathcal{T}_{\text{Reg}}$ を与えたときはより非線形回帰タスクにも対応可能なアルゴリズム $a_{\text{Reg}} \in \mathcal{A}$ が得られることが求められる.

以下, $\mathcal{T}$ に対して汎用的に性能が高いアルゴリズムのことを, 単に $\mathcal{T}$ に対する最適なアルゴリズムと表現する.

2.4 既存手法 RE-AutoML-Zero

本節では, Esteban らが提案した AutoML-Zero 手法について述べる. Esteban らが提案した手法は, Regularized Evolution (RE) を使った AutoML-Zero 手法であるため, 以降では RE-AutoML-Zero と呼ぶことにする. 以下, 第 2.4.1 項では RE-AutoML-Zero におけるアルゴリズムの表現方法, 第 2.4.2 項では RE-AutoML-Zero におけるアルゴリズムの評価方法, 第 2.4.3 項では RE による世代交代, 第 2.4.4 項では突然変異による個体生成について述べる.

2.4.1 アルゴリズムの表現方法

Code. 2.1 線型回帰アルゴリズムを AutoML-Zero の表現方法で表した例

```
1  def Setup():
2    s2 = 0.01 // 学習率の設定
3
4  def Predict():
5    s1 = dot(v1, v0) // 学習対象の重みと入力ベクトルの内積を計算
6
7  def Learn():
8    s3 = s0 - s1 // 予測ラベルと正解ラベルの誤差を計算
9    s3 = s2 * s3 // 学習率の適用
10   v2 = s3 * v0 // 重みの更新するための差分を計算
11   v1 = v1 + v2 // 重みを更新
```

RE-AutoML-Zero において機械学習アルゴリズムは, 小さな仮想メモリで動作するプログラムとして表される. 仮想メモリには, スカラー, $d^{(i)}$ 次元ベクトル, $d^{(i)} \times d^{(i)}$ 次元行列を複数個格納できる. ここで, $d^{(i)}$ はタスク集合 $\mathcal{T}_{\text{search}}$ に含まれるタスク $T^{(i)}$ の入力ベクトルの次元である. 以降, スカラーを格納する変数を $s1, s2, \cdots$, ベクトルを格納する変数を $v1, v2, \cdots$, 行列を格納する変数を $m1, m2, \cdots$ と表す. $s0$, $s1$, $v1$ は, それぞれ正解ラベル, アルゴリズムによる予測ラベル, 入力ベクトルを格納する先として使われる特別な変数である. その他の変数は学習対象のパラメータを格納したり, 計算結果を一時的に保存する用途で用いられる. 変数の個数の上限はスカラー, ベクトル, 行列それぞれに対してユーザが指定する必要がある.

アルゴリズムは, Code. 2.1 に示したように, Setup, Predict, Learn の 3 つの関数で表現される. 各関数は Table.A.1 に示した 64 個の命令の列で構成される. 命令は, 人間のバイアスを与えすぎないようにするため, 高校数学で学ぶ程度の演算のみを使用し, 機械学習のアルゴリズムの概念や行列の分解等の演算は含まれていない. また, 命令に与える引数は, 基本的には仮想メモリに格納されているスカラー $s1, s2, \cdots$, ベクトル $v1, v2, \cdots$, 行列 $m1, m2, \cdots$ のいずれかである. 一部例外として, 正規分布による乱数生成等の一部の命令では, μ, σ 等の定数が入力されることがある.

2.4.2 アルゴリズムの評価方法

[tbp] **Algorithm 1** タスク $T^{(i)}$ に対するアルゴリズムの評価方法

Input: (Setup, Predict, Learn): 評価対象のアルゴリズム, $\left(D^{(i)}_{\text{train}}, D^{(i)}_{\text{valid}}\right)$: タスク $T^{(i)}$ の学習用データと検証用データ

Output: タスク $T^{(i)}$ に対するアルゴリズムの評価値

```
1: initialize_memory()
2: Setup()
3: for e = 0; e < N_epochs; e++ do
4:    for all (x_j^(i), y_j^(i)) ∈ D_train^(i) do
5:       v0 ← x_j^(i)
6:       Predict()
7:       s1 ← Normalize(s1)
8:       s0 ← y_j^(i)
9:       Learn()
10:   end for
11: end for
12: l_sum = 0.0
13: for (x_j^(i), y_j^(i)) ∈ D_valid^(i) do
14:   v0 ← x_j^(i)
15:   Predict()
16:   s1 ← Normalize(s1)
17:   l_sum ← l_sum + Loss(y, s1)
18: end for
19: l_mean ← l_sum / |D_valid|
20: fitness = Rescale(l_mean)
21: return fitness
```

タスク集合内の 1 つのタスク $T^{(i)}$ に対するアルゴリズムの評価は, Algorithm 1 に示した流れで行われる. Algorithm 1 の入力 (Input) は評価対象のアルゴリズム, タスク $T^{(i)} \in \mathcal{T}_{\text{search}}$ の学習用データ $D^{(i)}_{\text{train}}$ および検証用データ $D^{(i)}_{\text{valid}}$ であり, 出力 (Output) は評価対象のアルゴリズムのタスク $T^{(i)}$ に対する評価値である. Algorithm 1 の各行の説明を以下に示す.

1 行目 メモリをすべて 0 で初期化する.

2 行目 Setup 関数を実行する.

3-11 行目 学習用のループを実行する. N_{epochs} はエポック数であり, 学習用データを使う回数を表す.

4-10 **行目**　e 番目のエポックに対応する学習を行う．エポックごと $(\boldsymbol{x}_j^{(i)}, y_j^{(i)}) \in D_{\text{train}}^{(i)}$ を取り出す順番は異なる．

5 **行目**　$v0$ に入力ベクトル $\boldsymbol{x}_j^{(i)}$ を代入する．

6 **行目**　Predict 関数の実行を行う．正解ラベル $y_j^{(i)}$ は事前に代入されていないので使うことが出来ない．また, Predict 関数実行後は, $s1$ に予測結果が含まれているものとして扱う．

7 **行目**　$s1$ に格納されている予測結果を Normalize 関数を用いて正規化する．Normalize 関数は, 回帰タスクの場合には恒等関数, 二値分類タスクの場合には sigmoid 関数が使われる．

8 **行目**　正解ラベルを $s0$ に代入する．

9 **行目**　Learn 関数を実行する．

12 **行目**　検証用データにおける損失の合計を計算するための変数 l_{sum} を初期化する．

13-18 **行目**　検証用データ $D_{\text{valid}}^{(i)}$ を用いて検証用ループを実行する．

14 **行目**　学習用ループと同様に $v0$ に入力ベクトル $\boldsymbol{x}_j^{(i)}$ を代入する．

15 **行目**　学習用ループと同様に Predict 関数の実行を行う．

16 **行目**　学習用ループと同様に予測結果を正規化する．

17 **行目**　学習用ループとは異なり, Learn 関数の実行はせず予測結果の損失を Loss 関数を使って計算する．Loss 関数は, 回帰タスクの場合には二乗誤差を返す関数, 二値分類タスクの場合にはの場合には予測ラベルと正解ラベルが一致しているときに 1, それ以外のときは 0 を返す関数である．

19 **行目**　損失の平均値 l_{mean} を計算する．

20 **行目**　損失を Rescale 関数を用いて評価値に変換する．評価値の変域は $[0, 1]$ で 1 に近いほど, 損失が小さいことに対応する．Rescale 関数は, 回帰タスクの場合は $\text{Rescale}(l) = 1 - \frac{2}{\pi} \arctan \sqrt{l}$, 二値分類タスクの場合は $\text{Rescale}(l) = 1 - l$ である．

21 **行目**　評価値を返却する．

1 行目で行ったメモリの初期化以降, 特別な変数 $s0$, $s1$, $v0$ 以外の変数への代入は, Setup, Predict, Learn の中以外で行われることがない．そのため, 評価対象のアルゴリズムの各関数 Setup, Predict, Learn では, $s0$, $s1$, $v0$ 以外の変数に学習対象のパラメータを格納することで, 初期化時から検証時まで当該パラメータの値を引き継ぐことができる．

タスク集合 $\mathcal{T}_{\text{search}}$ に対するアルゴリズムの評価値は, タスク集合 $\mathcal{T}_{\text{search}}$ 内の各タスク $T^{(i)}$ に対して Algorithm 1 で評価を行った結果の平均値となる．したがって, 評価値が高いアルゴリズムは $\mathcal{T}_{\text{search}}$ に含まれるタスクを汎用的かつ高性能で解けるアルゴリズムとなる．$\mathcal{T}_{\text{search}}$ の構成方法が偏っていなければ, $\mathcal{T}_{\text{search}}$ に対する評価値が高いアルゴリズムは, $\mathcal{T}_{\text{eval}}$ や $\mathcal{T}$ のタスクに対しても汎用的かつ高性能に解くことが出来ると考えられる．

2.4.3　RE による世代交代

Esteban らが提案した RE-AutoML-Zero では, N_{pop} 個のアルゴリズムをランダムに初期集団として生成した後に, Fig.2.1 の STEP1 から STEP4 に示した Regularized Evolution (RE) を繰り返し行うことで, 最適なアルゴリズムの探索を行う．RE では, STEP1 で最も古い個体を削除した後に, STEP2 でトーナメント選択, すなわち $K(< N_{\text{pop}})$ 個の個体を非復元抽出した上で最も評価値の高い個体を選択を行う．その後, STEP3 でトーナメント選択によって選ばれた個体をコピーし, STEP4 で一定確率 p_{mutate} で突然変異を行う．集団サイズ N_{pop}, トーナメントサイズ K, 突然変異確率 p_{mutate} はユーザパラメータである．

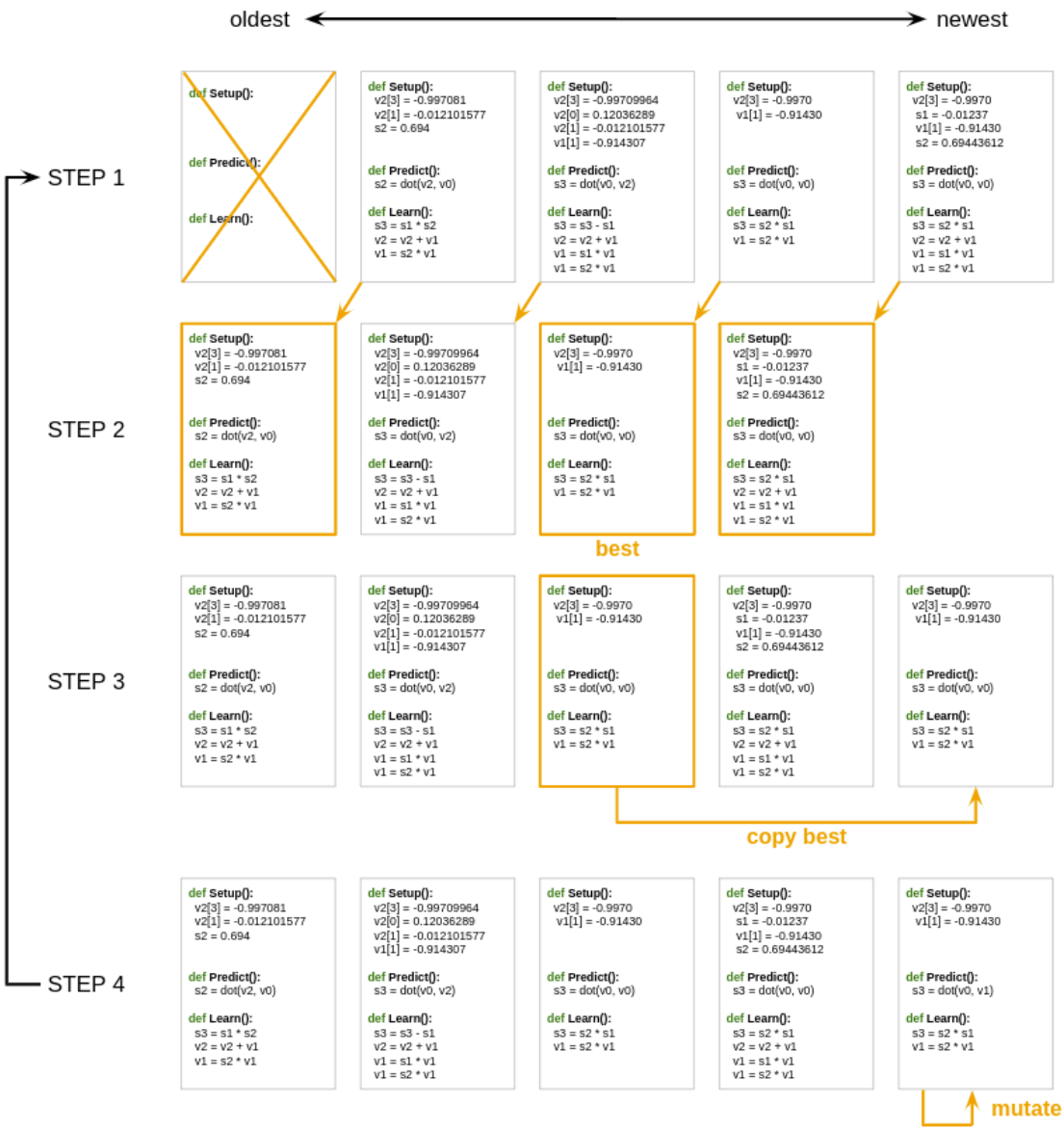

Fig.2.1　RE-AutoML-Zero[11] の世代交代モデル．STEP1 から STEP4 を繰り返すことで最適なアルゴリズムの発見を目指す．STEP1 で最も古い個体を削除した後に, STEP2 でトーナメント選択を行う．その後, STEP3 でトーナメント選択によって選ばれた個体をコピーし, STEP4 で一定確率 p_{mutate} で突然変異を行う．

[tbp] **Algorithm 2** RE-AutoML-Zero のアルゴリズム

Input: タスク集合 $\mathcal{T}_{\text{search}}$, 集団サイズ N_{pop}, トーナメントサイズ K, 突然変異確率 p_{mutate}, 最大評価回数 N_{eval}

Output: 探索結果のアルゴリズム

1: $(P,\ \text{best}) \leftarrow \text{initialize}(N_{\text{pop}})$
2: $\text{eval\_num} \leftarrow N_{\text{pop}}$
3: **while** $\text{eval\_num} < N_{\text{eval}}$ **do**

```
    remove_oldest(P)
    a = tournament_select(P, K)
    if rand(0, 1) < p_mutate then
      mutate(a)
      algorithm.fitness ← evaluate(a, T_search)
      eval_num ← eval_num + 1
      if algorithm.fitness > best.fitness then
        best ← a
      end if
    end if
    add(P, a)
  end while
  return best
```

RE-AutoML-Zero の詳細なアルゴリズムを Algorithm 2 に示す．Algorithm 2 の入力 (Input) はタスク集合 $\mathcal{T}_{\text{search}}$, 集団サイズ N_{pop}, トーナメントサイズ K, 突然変異確率 p_{mutate}, 最大評価回数 N_{eval} である．また, 出力 (Output) は探索結果のアルゴリズムである．Algorithm 2 の各行の説明を以下に示す．

1 行目　initialize を用いて, アルゴリズム (個体) をランダムに N_{pop} 個生成した上で, 各アルゴリズムを評価を行う．initialize(N_{pop}) の返り値は, 生成された初期集団と初期集団内の評価値が最上位のアルゴリズム (個体) である．初期集団内の個体は, 第 2.4.4 項で述べる突然変異手法 (2) を Setup, Predict, Learn すべてに適用することで生成する．このとき, 各関数の命令数は事前にユーザパラメータとして与える．

2 行目　評価回数のカウンターを初期集団生成分の N_{pop} で初期化する．

3-15 行目　評価回数が上限を超えるまで RE による世代交代を繰り返す．

- **4 行目**　集団 P から最も古いアルゴリズム (個体) を除去する関数 remove_oldest(P) を実行する．
- **5 行目**　集団 P からトーナメントサイズ K でトーナメント選択する関数 tournament_select(P, K) を実行する．tournament_select の返り値は, 選択されたアルゴリズムのコピーである．
- **6-13 行目**　一定確率 p_{mutate} で突然変異を行い, 評価値が改善されたら best に反映する．ここで, rand$(0, 1)$: 0 から 1 の範囲の実数の一様乱数を表す．
  - **7 行目**　アルゴリズム a を突然変異させる関数 mutate(a) を実行する．突然変異方法の詳細は第 2.4.4 項で述べる．
  - **8 行目**　タスク集合 $\mathcal{T}_{\text{search}}$ に対するアルゴリズム a の評価値を第 2.4.2 項に示した方法で計算する関数 evaluate($\mathcal{T}_{\text{search}}, a$) を実行し, 個体に記録する．
  - **9 行目**　評価回数をインクリメントする．
  - **10-12 行目**　突然変異した結果のアルゴリズム a の評価値が現状の best を超える場合は best に a を格納する．
- **14 行目**　集団 P にアルゴリズム a を追加する関数 add(P, a) を実行する．

16 行目　best を返却する．

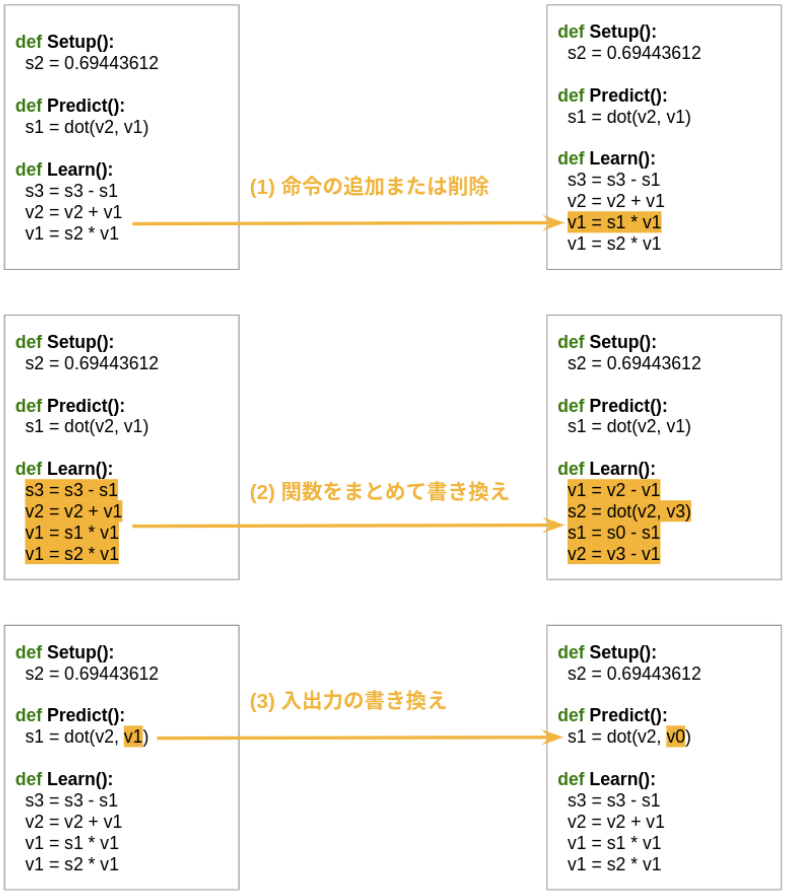

Fig.2.2　突然変異の種類 [11]. (1) アルゴリズムの Setup, Predict, Learn のいずれかをランダムに選択した上で, ランダムに命令を追加または削除する. (2) アルゴリズムの Setup, Predict, Learn のいずれかをすべてランダムな命令列で書き換える. (3) アルゴリズム内のいずれかの命令の入出力をランダムに変更する.

2.4.4　突然変異による個体生成

RE-AutoML-Zero の突然変異は図 2.2 に示した 3 種類が存在する. それぞれの突然変異手法の以下で説明する.

(1) アルゴリズムの構成要素 (Setup, Predict, Learn) のいずれかを一様ランダムに選択した上で, 選択された構成要素に対して命令の追加または削除を行う突然変異. 追加または削除する場所も一様ランダムに決まる.
(2) アルゴリズムの構成要素（Setup, Predict, Learn）のいずれかを, ランダムな命令列で書き換える突然変異. 命令数は元の個数と変更しない.
(3) アルゴリズムの構成要素（Setup, Predict, Learn）のいずれかを選択した上で, その構成要素内の 1 つの命令の入力, 出力もしくは即値のいずれかをランダムに変更する突然変異.

突然変異はユーザが与える制限を満たす範囲でのみ行うことができる. 具体的には, ユーザは各構成要素に含まれる命令数の下限および上限, 各構成要素に含められる命令の種類, 仮想メモリ上の

変数の個数を指定することができる. そのため, (1) によってユーザが設定した構成要素の命令数の上限を超えた追加や下限を下回る削除を行ったり, (2) によって許可されていない命令を構成要素に含めることは出来ない. また, 仮想メモリ上のスカラーの変数の個数が 3 個と決まっている場合, すなわち $s0$ から $s2$ まで使える場合に, (3) で命令の入力変数として $s4$ に設定することはできない.

2.5 おわりに

本章では, 本研究で対象とする問題設定を明確にした上で, 既存手法である Esteban らの手法 RE-AutoML-Zero におけるアルゴリズムの表現方法やアルゴリズムの評価方法, 世代交代モデル, 突然変異手法について説明した.

第 3 章

MGG-AutoML-Zero+AV

3.1 はじめに

本章では, 既存手法である Esteban らの手法 RE-AutoML-Zero の問題点を指摘した上で, その問題点に対処した手法を提案する. 既存手法には世代交代モデルに関する問題点と探索空間の冗長性に関する問題があると考えられる. 提案手法では, 新たな世代交代モデル Minimal Generation Gap (MGG) を導入した上で, 妥当であるかを検証する処理 (Algorithm Validation, AV) を行うことでこれらの問題点に対処する. 以下, 第 3.2 節では RE-AutoML-Zero の問題点の詳細, 第 3.3 節では問題点対処した提案手法を説明する. 本章では, 既存手法は RE-AutoML-Zero, 提案手法は MGG-AutoML-Zero+AV を指す.

3.2 RE-AutoML-Zero の問題点

既存手法の RE-AutoML-Zero には, 集団の多様維持および探索空間の冗長性に関する問題が存在すると考えられる. 以下では, それぞれの問題点の詳細を述べる.

3.2.1 集団の多様維持に関する問題

既存手法の RE-AutoML-Zero で採用されている世代交代モデルである RE は, 集団の多様性を失いやすいと考えられる. 佐藤らの論文 [12] の考察を踏まえると, RE が集団の多様性を低下させる要因として, 主に以下の 3 点が考えられる.

1. Fig.2.1 の STEP1 (Algorithm 2 の 17 行目) において, 無条件で元の集団の個体が淘汰されている点
2. Fig.2.1 の STEP2 (Algorithm 2 の 18 行目) における複製選択で, 強い選択圧が掛かるトーナメント選択が用いられている点
3. Fig.2.1 の STEP3 および STEP4(Algorithm 2 の 19 行目から 29 行目) における生存選択で選ばれる個体が淘汰される個体と無関係である点

1 つめは, 希少な構造を持つ個体も無条件で淘汰されることを意味しており, 多様性の低下を招くと考えられる. 2 つめは, 探索序盤で得られた局所最適解に対しても選択圧が強く掛かることを意味しており, 十分な探索をする前に多様性を低下させる恐れがある. 3 つめは, 淘汰される個体の形質が次世代に引き継げなくなることを意味しており, 集団の多様性を低下させる要因になると考えられる. これらの要因が引き起こす多様性の低下によって, 探索までに要す評価回数が増加したり, 局所最適解に陥りやすくなることが佐藤らの論文 [12] でも示されている.

3.2.2 探索空間の冗長性に関する問題

既存手法 RE-AutoML-Zero では, 妥当ではない機械学習アルゴリズムも探索対象としているため, 探索空間が冗長であり探索が非効率的になっていると考えられる. 以下では, 本論文における妥当な機械学習アルゴリズムの定義を述べた上で, 既存手法 RE-AutoML-Zero で探索対象となる妥当ではない機械学習アルゴリズムの例を挙げ, 探索空間に冗長性があることを説明する.

本論文において妥当な機械学習アルゴリズムとは, 以下の性質を有しているアルゴリズムのことをいう.

- Predict 関数の最後が $s1$ への代入命令である.
- 予測ラベル $s1$ に関する性質.
  - 直前で代入された入力ベクトル $v0$ に依存している.
  - 学習対象のパラメータに依存している.
- 学習対象のパラメータに関する性質
  - 更新される前の自分自身に依存している.
  - 直前に代入された正解ラベル $s0$ に依存している
  - 直前で代入された予測ラベル $s1$ に依存している.
  - 直前で代入された入力ベクトル $v0$ に依存している

妥当な機械学習アルゴリズムの例を Code. 3.1 に示す. Code. 3.1 において学習対象のパラメータは $v1$ のことである. 一般には, 学習対象のパラメータが格納される変数が $v1$ 以外であったり, 複数使われる場合が想定される. その場合であっても, 妥当な機械学習アルゴリズムは $v1$ と同じ役割をする学習対象のパラメータを少なくとも 1 つは持つものとする.

既存手法 RE-AutoML-Zero では, 多種多様な妥当ではない機械学習アルゴリズムが探索対象となる. RE-AutoML-Zero で探索対象となる妥当ではない機械アルゴリズムの具体例を Code. 3.2 から 3.8 に示す. また, Table.3.1 に Code. 3.2 から 3.8 のアルゴリズムが妥当ではない理由を示す.

妥当な機械学習アルゴリズムに要求される性質は, 機械学習タスクを高性能に解く上でタスクの種類に関係なく必要な性質であるため, 妥当ではない機械学習アルゴリズムが探索対象となっている既存手法の探索空間は冗長であると考えられる. RE-AutoML-Zero におけるアルゴリズム評価は, 第 2.4.1 項に示したようにタスク集合 $\mathcal{T}_{\text{search}}$ に含まれるすべてのタスクに対して学習と検証を行うため多くの時間を必要とする. そのため, 探索空間が冗長な既存手法では, 妥当ではない機械学習アルゴリズムに対して多くの評価が行われ, 探索効率を低下させていると考えられる.

Table.3.1　既存手法 RE-AutoML-Zero で探索対象としている妥当ではない機械学習アルゴリズムの例と妥当ではない理由

具体例	妥当な機械学習アルゴリズムではない理由
Code. 3.2	Predict 関数の最後の命令で予測ラベル $s1$ に代入が行われていない.
Code. 3.3	Predict 関数の中で代入される予測ラベル $s1$ の値が Predict 関数実行前に代入される入力ベクトル $v0$ に依存していない.
Code. 3.4	Predict 関数の中で代入される予測ラベル $s1$ が Learn 関数で更新されている学習対象のパラメータ $v1$ に依存していない.
Code. 3.5	Learn 関数において代入される学習対象のパラメータ $v1$ が代入前の自身に依存していない.
Code. 3.6	Learn 関数において代入される学習対象のパラメータ $v1$ が Learn 関数の実行前に代入される正解ラベル $s0$ に依存してない.
Code. 3.7	Learn 関数において代入される学習対象のパラメータ $v1$ が直前の Predict 関数で代入した予測ラベル $s1$ に依存していない.
Code. 3.8	Learn 関数において代入される学習対象のパラメータ $v1$ が直前の Predict 関数の実行前に代入される入力ベクトル $v0$ に依存していない.

Code. 3.1　妥当な機械学習アルゴリズムの例

```
1  def Setup():
2    s2 = 0.01
3
4  def Predict():
5    s1 = dot(v1, v0)
6
7  def Learn():
8    s3 = s2 - s1
9    s3 = s2 * s3
10   v2 = s3 * v0
11   v1 = v1 + v2
```

Code. 3.2　Predict 関数の中で予測ラベル $s1$ に代入が行われておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1  def Setup():
2    s2 = 0.01
3
4  def Predict():
5    s2 = dot(v1, v0)
6
7  def Learn():
8    s3 = s0 - s1
9    s3 = s2 * s3
10   v2 = s3 * v0
11   v1 = v1 + v2
```

Code. 3.3　Predict 関数の中で代入される予測ラベル $s1$ の値が Predict 関数実行前に代入される入力ベクトル $v0$ に依存しておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1     def Setup():
2       s2 = 0.01
3
4     def Predict():
5       s1 = dot(v1, v3)
6
7     def Learn():
8       s3 = s0 - s1
9       s3 = s2 * s3
10      v2 = s3 * v0
11      v1 = v1 + v2
```

Code. 3.4　Predict 関数の中で代入される予測ラベル $s1$ が Learn 関数で更新されている学習対象のパラメータ $v1$ に依存して依存しておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1     def Setup():
2       s2 = 0.01
3
4     def Predict():
5       s1 = dot(v3, v0)
6
7     def Learn():
8       s3 = s0 - s1
9       s3 = s2 * s3
10      v2 = s3 * v0
11      v1 = v1 + v2
```

Code. 3.5　Learn 関数において代入される学習対象のパラメータ $v1$ が代入前の自身に依存しておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1     def Setup():
2       s2 = 0.01
3
4     def Predict():
5       s1 = dot(v1, v0)
6
7     def Learn():
8       s3 = s0 - s1
9       s3 = s2 * s3
10      v2 = s3 * v0
11      v1 = v0 + v2
```

Code. 3.6　Learn 関数において代入される学習対象のパラメータ $v1$ が Learn 関数の実行前に代入される正解ラベル $s0$ に依存しておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1     def Setup():
2       s2 = 0.01
3
4     def Predict():
5       s1 = dot(v1, v0)
6
7     def Learn():
8       s3 = s2 - s1
9       s3 = s2 * s3
10      v2 = s3 * v0
11      v1 = v1 + v2
```

Code. 3.7　Learn 関数において代入される学習対象のパラメータ $v1$ が直前の Predict 関数で代入した予測ラベル $s1$ に依存しておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1     def Setup():
2       s2 = 0.01
3
4     def Predict():
5       s1 = dot(v1, v0)
6
7     def Learn():
8       s3 = s0 - s2
9       s3 = s2 * s3
10      v2 = s3 * v0
11      v1 = v1 + v2
```

Code. 3.8　Learn 関数において代入される学習対象のパラメータ $v1$ が直前の Predict 関数の実行前に代入される入力ベクトル $v0$ に依存しておらず 妥当ではないアルゴリズムの例. 妥当なアルゴリズム Code. 3.1 との差分を赤字で示す.

```
1     def Setup():
2       s2 = 0.01
3
4     def Predict():
5       s1 = dot(v1, v0)
6
7     def Learn():
8       s3 = s0 - s1
9       s3 = s2 * s3
10      v2 = s3 * v1
11      v1 = v1 + v2
```

3.3　提案手法

本章では, 既存手法である Esteban らの手法の問題点に対処した手法を提案する. 提案手法では, 既存手法の RE-AutoML-Zero を大きく分けて 2 つの点で見直した.

1 つめは世代交代モデルである. 第 3.2.1 項で述べたように, RE-AutoML-Zero で用いられている RE は集団の多様性を失いやすい世代交代モデルであった. そこで本研究では, 佐藤らが論文 [12] の中で提案した世代交代モデル Minimal Generation Gap (MGG) を導入することで, 集団の多様性を維持しやすくなるように工夫する. 以降, RE-AutoML-Zero の世代交代モデルを MGG に変更した手法を MGG-AutoML-Zero と呼ぶ.

2 つめは探索空間である. 第 3.2.2 項で述べたように, RE-AutoML-Zero では妥当ではない機械学習アルゴリズムが探索空間に多く含まれていた. そこで我々は, 機械学習アルゴリズムとして妥当であるかを検証する処理 (Algorithm Validation, AV) をアルゴリズムの生成の度に行うことで, 妥当ではない機械学習アルゴリズムを実行不可能解とする手法を提案する. 以降, MGG-AutoML-Zero に AV を加えた手法を MGG-AutoML-Zero+AV と呼ぶ.

以下, 第 3.3.1 項では MGG による集団の多様性維持の基本的な考え方と詳細, 第 3.3.2 項では AV による探索空間の削減の基本的な考え方と詳細を述べる.

3.3.1　MGG による集団の多様性維持

本節では, 3.3.1 で MGG による集団の多様性維持の基本的な考え方を述べた後で, 3.3.1 で MGG-AutoML-Zero のアルゴリズムの詳細を述べる.

MGG-AutoML-Zero の基本的な考え方

MGG-AutoML-Zero では, RE-AutoML-Zero と同様の方法で初期集団を生成した後に, Fig.3.1 に示した MGG による世代交代を繰り返し行う. 各世代交代では, STEP1 で無作為に 2 つのアルゴリズム p_a, p_b を親個体として取り出し, STEP2 で p_a を突然変異させた個体を

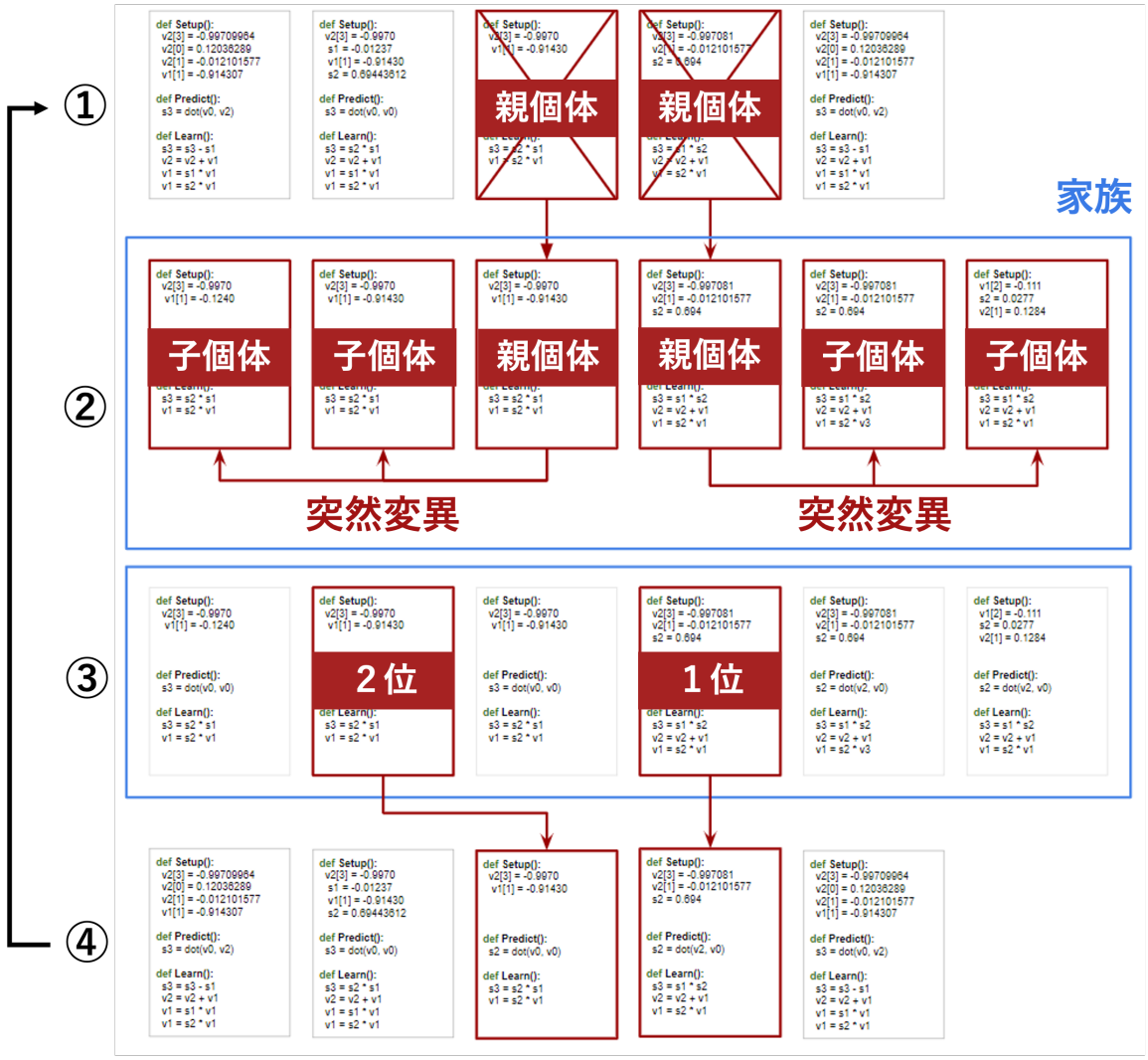

Fig.3.1　MGG-AutoML-Zero の世代交代モデル．STEP1 から STEP4 を繰り返す．STEP1 で無作為に 2 つのアルゴリズム p_a, p_b を親個体として取り出し，STEP2 で p_a を突然変異させた個体を $N_{\text{children}}/2$ 個，p_b を突然変異させた個体を $N_{\text{children}}/2$ 個つくり，合計で N_{children} 個の子個体を生成する．その後の STEP3,4 では，生成した N_{children} 個の子個体と親個体 p_a, p_b の中で最も評価値が高い上位 2 つの個体を集団に戻す．

$N_{\text{children}}/2$ 個，p_b を突然変異させた個体を $N_{\text{children}}/2$ 個つくり，合計で N_{children} 個の子個体を生成する．その後の STEP3,4 では，生成した N_{children} 個の子個体と親個体 p_a, p_b の中で最も評価値が高い上位 2 つの個体を集団に戻す．集団サイズ N_{pop}，子個体生成数 N_{children} はユーザパラメータである．

RE-AutoML-Zero から MGG-AutoML-Zero になることで，第 3.2.1 項で述べた 3 つの集団の多様性を低下させる要因に対処できると考えられる．以下に，集団の多様性を低下させる 3 つの要因の内容を再掲し，MGG-AutoML-Zero にすることで対処可能になる理由を示す．

多様性低下要因 1　(内容)　RE-AutoML-Zero では，Fig.2.1 の STEP1 (Algorithm 2 の 17 行目) において，無条件で元の集団の個体が淘汰されている．

(対処可能な理由)　MGG-AutoML-Zero では，Fig.3.1 の STEP1 で淘汰される候補となっ

た親個体も, Fig.3.1 の STEP3, 4 の生存選択で集団に戻される可能性があるため, 集団内の個体が無条件で淘汰されることはない.

多様性低下要因 2　(内容)　RE-AutoML-Zero では, Fig.2.1 の STEP2 (Algorithm 2 の 18 行目) における複製選択で, 強い選択圧が掛かるトーナメント選択が用いられている.

(対処可能な理由)　MGG-AutoML-Zero では, Fig.3.1 の STEP1 で無作為に複製選択するため, 複製選択で強い選択圧がかかることはない.

多様性低下要因 3　(内容)　RE-AutoML-Zero では, Fig.2.1 の STEP3 および STEP4(Algorithm 2 の 19 行目から 29 行目) における生存選択で選ばれる個体が淘汰される個体と無関係である.

(対処可能な理由)　MGG-AutoML-Zero では, Fig.3.1 の STEP3, 4 における生存選択で選ばれる個体が, 淘汰される候補である親個体の家族に限定されている.

佐藤らの論文 [12] の MGG における生存選択は, 子個体と親個体の中から最上位の個体とルーレット選択で選ばれた個体を取り出すことで行っていたが, 本研究の MGG の生存選択では, 最も評価値が高い上位 2 つを取り出すことで行う. 佐藤らの論文 [12] の対象問題では, 生成される子個体の評価値が親個体よりも改善される可能性が高い. 一方, 本研究で生成される子個体は突然変異オペレータのランダム性が高く, 親個体より評価値が高い子個体が得られる可能性が低い. 故に, ルーレット選択をしてしまうと, 評価値が悪い個体が数によって選ばれてしまい, 進化が滞ってしまうと考えられる.

MGG-AutoML-Zero の詳細なアルゴリズム

MGG-AutoML-Zero の詳細なアルゴリズムを Algorithm 3 に示す. Algorithm 3 の入力 (Input) はタスク集合 $\mathcal{T}_{\text{search}}$, 集団サイズ N_{pop}, 子個体生成数 N_{children}, 最大評価回数 N_{eval} である. また, 出力 (Output) は探索結果のアルゴリズムである. Algorithm 3 の各行の説明を以下に示す.

1 行目　RE-AutoML-Zero と同様の方法で初期集団を生成して評価を行う.

2 行目　評価回数のカウンターを初期集団生成分の N_{pop} で初期化する.

3-41 行目　評価回数が上限を超えるまで MGG による世代交代を繰り返す.

- **4-5 行目**　集団 P からランダムにアルゴリズム (個体) を非復元抽出する関数 randomly_remove(P) を用いて親個体を 2 つ選択する.
- **6-12 行目**　2 つの親子体のうち評価値が高い方を best1, 低い方を best2 に設定する. best1 と best2 は, 最後に集団に戻す対象となるアルゴリズムを格納する用の変数である.
- **13 行目**　子個体の生成 index を 0 で初期化する.
- **14-35 行目**　子個体を N_{children} 個生成する用のループを実行する.
  - **15-19 行目**　child_index $< N_{\text{children}}/2$ の場合は child に parent1 をコピーし, それ以外の場合は child に parent2 をコピーする. これにより, それぞれの親個体から $N_{\text{children}}/2$ 個ずつ個体を生成することが可能となる.
  - **20 行目**　親個体いずれかのコピーである child を突然変異させる. 突然変異に使う関数 mutate は RE-AutoML-Zero と同じである.
  - **21 行目**　RE-AutoML-Zero と同様の方法で, $\mathcal{T}_{\text{search}}$ に対するアルゴリズム child の評価を行う.
  - **22 行目**　評価回数をインクリメントする.
  - **23-30 行目**　子個体 child の評価値が best2 以上である場合は, 適切に順位を更新する.
  - **31-33 行目**　この時点で評価回数が上限を超えた場合は, 子個体が所定の個数生成されていなくても強制的に子個体生成のループから抜ける.

34 **行目**　子個体の生成 index をインクリメントする.

36-37 **行目**　集団 P にアルゴリズム a を追加する関数 add(P, a) を用いて, best1 と best2 を集団に追加する.

38-40 **行目**　アルゴリズム best1 の評価値が現状の best を超える場合は best に best1 を格納する

42 **行目**　best を返却する.

Algorithm 3 AutoML-Zero+MGG のアルゴリズム

Input: 集団サイズ N_{pop}, 子個体生成数 N_{children}, 最大評価回数 N_{eval}
Output: 探索結果のアルゴリズム

```
(P, best) ← initialize(N_pop)
eval_num ← N_pop
while eval_num < N_eval do
    parent1 = randomly_remove(P)
    parent2 = randomly_remove(P)
    if parent1.fitness > parent2.fitness then
        best1 ← parent1
        best2 ← parent2
    else
        best1 ← parent2
        best2 ← parent1
    end if
    child_index ← 0
    while child_index < N_children do
        if child_index < N_children/2 then
            child ← copy(parent1)
        else
            child ← copy(parent2)
        end if
        mutate(child)
        child.fitness ← evaluate(child)
        eval_num ← eval_num + 1
        if child.fitness ≥ best2.fitness then
            if child.fitness ≥ best1.fitness then
                best2 ← best1
                best1 ← child
            else
                best2 ← child
            end if
        end if
        if eval_num ≥ N_eval then
            break
        end if
        child_index ← child_index + 1
    end while
```

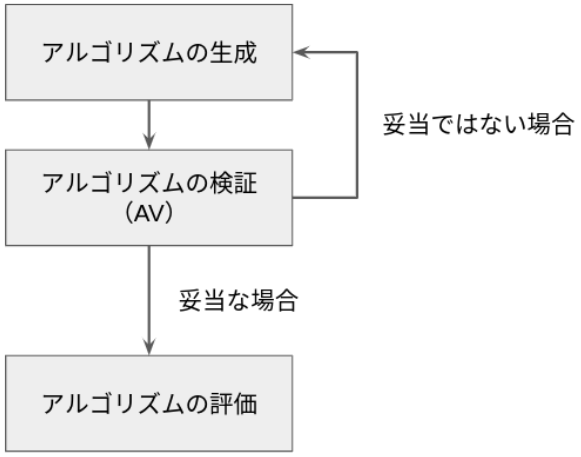

Fig.3.2　アルゴリズムの検証を導入した場合のアルゴリズムの生成から評価までの流れ. 妥当なアルゴリズムが生成されるまで生成を繰り返すことで, 妥当なアルゴリズム以外は評価が行われない.

add(P, best1)
add(P, best2)
if best1.fitness > best.fitness **then**
best ← best1
end if
end while
return best

3.3.2　AV による探索空間の削減

本節では, AV による探索空間の削減の基本的な考え方, AV を導入するにあたって必要となる依存関係解析, AV の詳細なアルゴリズムを述べる. 以下, 3.3.2 では AV の基本的な考え方, 3.3.2 では依存関係解析の詳細なアルゴリズム, 3.3.2 では AV の詳細なアルゴリズムについて述べる.

AV の基本的な考え方

アルゴリズムの検証 (AV) を導入した手法では, Fig.3.2 に示したように妥当な機械学習アルゴリズム (個体) が生成されるまで個体の生成を繰り返す. これによって, 妥当ではない機械学習アルゴリズムに対して, 時間のかかるアルゴリズムの評価を行わず, 実行不可能解とすることが可能となる. MGG-AutoML-Zero+AV の場合は, 初期集団の各個体を生成する時および突然変異で子個体を生成する時に Fig.3.2 の処理が行われることになる. ただし, 妥当ではないアルゴリズムは集団サイズや子個体の生成数には含めない. つまり MGG-AutoML-Zero+AV において, 集団は N_{pop} 個の妥当な機械学習アルゴリズムであり, 子個体生成では毎回 N_{children} 個の妥当な機械学習アルゴリズムが生成される.

AV では, 機械学習アルゴリズムとして妥当性を以下の検証によって確認する.

- Predict 関数の最後が s1 への代入命令であることの検証 (Predict Function Validation, PFV).

Table.3.2　AV によって実行不可能解になる具体例と制約を受ける検証の名称. 制約を受ける検証の名称は複数存在しても 1 つしか記載していない.

具体例	制約を受ける検証の名称
Code. 3.2	PFV
Code. 3.3	PLV_{v0}
Code. 3.4	$\mathrm{PLV}_{\mathrm{LP}}$
Code. 3.5	$\mathrm{LPV}_{\mathrm{prev}}$
Code. 3.6	LPV_{v0}
Code. 3.7	LPV_{s1}
Code. 3.8	LPV_{v0}

- 予測ラベル $s1$ の依存関係の検証 (Prediction Label Validation, PLV)
  - PLV_{v0}: 直前で代入された入力ベクトル $v0$ に依存していることの検証.
  - $\mathrm{PLV}_{\mathrm{LP}}$: 学習対象のパラメータに依存していることの検証.
- 学習対象のパラメータの依存関係の検証 (Learning Parameter Validation, LPV)
  - $\mathrm{LPV}_{\mathrm{prev}}$: 更新される前の自分自身に依存していることの検証.
  - LPV_{s1}: 直前に代入された正解ラベル $s0$ に依存していることの検証.
  - LPV_{s0}: 直前で代入された予測ラベル $s1$ に依存していることの検証.
  - LPV_{v0}: 直前で代入された入力ベクトル $v0$ に依存していることの検証. ただし, $s1$ を介した依存は含まれない.

各検証はすべて第 3.2.2 項で述べた妥当な機械学習アルゴリズムが持つ性質と 1 対 1 に対応している. AV の導入によって, 妥当ではない機械学習アルゴリズムの例として示した Code. 3.2 から Code. 3.8 はすべて実行不可能解となる. 具体的には, Table.3.2 に示した対応関係で実行不可能解になる.

依存関係解析の詳細なアルゴリズム

本項では, AV の導入にあたって必要となる依存関係解析の詳細なアルゴリズムについて説明する. 依存関係解析では, 命令列 $F = (\mathrm{instr}_1, \mathrm{instr}_2, \cdots, \mathrm{instr}_n)$ と解析対象の変数の集合 $V = \{\mathrm{var1}, \mathrm{var2}, \cdots\}$ が与えられた時に, F 実行後の $\mathrm{var1}, \mathrm{var2}, \cdots$ の値に影響を与える F 実行前の変数の集合 $D(F, V)$ を求める. ここで, F は Setup, Predict, Learn 等の命令列であり, V にはスカラー変数 $s1, s2, \cdots$, ベクトル変数 $v1, v2, \cdots$, 行列変数 $m1, m2, \cdots$ の一部が要素として含まれる. 例えば, Code. 3.1 においては, $D\,(\mathrm{Learn}, \{v1\}) = \{s0,\ s1,\ s2,\ v0,\ v1\}$ である. 実際, Learn 実行前における $s0,\ s1,\ s2,\ v0,\ v1$ のいずれかの値が変わると, Learn 実行後の $v1$ の値が変化する. これに対し, Learn 実行前における $s0,\ s1,\ s2,\ v0,\ v1$ 以外の変数の値, 例えば $v2$ が変わったとしても, Learn 実行後の $v1$ の値は変化ない.

依存関係解析は命令列を後ろから解析していくことで行うことができる. 実際, $D_i = D((\mathrm{instr}_i, \mathrm{instr}_{i+1}, \cdots, \mathrm{instr}_n), V)$, $D_{n+1} = V$ とすれば, D_i は $1 \leq i \leq n$ の範囲で次の漸化式を満たす.

$$D_i = \begin{cases} D_i = D_{i+1} & \mathrm{output}(\mathrm{instr}_i) \notin D_{i+1} \\ D_i = (D_{i+1} \setminus \mathrm{output}(\mathrm{instr}_i)) \cup \mathrm{input}(\mathrm{instr}_i) & \mathrm{output}(\mathrm{instr}_i) \in D_{i+1} \end{cases} \tag{3.1}$$

ここで, $\mathrm{output}(\mathrm{instr}_i)$ は命令 instr_i の出力変数を返す関数であり, $\mathrm{input}(\mathrm{instr}_i)$ は instr_i の入力

変数の集合を返す関数である.

式 3.1 の漸化式を用いた依存関係解析の詳細なアルゴリズムを Algorithm 4 に示す. Algorithm 4 の入力 (Input) は命令列 F と解析対象の変数の集合 V であり, 出力 (Output) は $D(F,V)$ である. Algorithm 4 の各行の説明を以下に示す.

1 行目 D_i を $D_{n+1}=V$ で初期化する.
2 行目 i を n で初期化する.
3-10 行目 命令列の後ろから順番に解析を行う. 各ループの中では D_i を漸化式に基づいて更新した上で, i をデクリメントする.
4 行目 instr_i の出力先変数を取得し, o に格納する.
5-8 行目 式 3.1 の漸化式に従って更新をする.
9 行目 i をデクリメントする.
11 行目 $D_1=D(F,V)$ を返却する.

Algorithm 4 依存関係解析のアルゴリズム

Input: $F=(\mathrm{instr}_1,\mathrm{instr}_2,\cdots,\mathrm{instr}_n)$: 命令列, $V=\{\mathrm{var1},\mathrm{var2},\cdots\}$: 解析対象の変数の集合
Output: $D\,(F,V)$: 命令列 F 実行後に $\mathrm{var1},\mathrm{var2},\cdots$ に格納されている値が依存している F 実行前の変数の集合

1: $D_i \leftarrow V$
2: $i \leftarrow n$
3: **while** $i>0$ **do**
4: $\quad o \leftarrow \mathrm{output}(\mathrm{instr}_i)$
5: $\quad$ **if** $o \in D_i$ **then**
6: $\quad\quad D_i \leftarrow D_i \setminus \{o\}$
7: $\quad\quad D_i \leftarrow D_i \cup \mathrm{input}(\mathrm{instr}_i)$
8: $\quad$ **end if**
9: $\quad i \leftarrow i-1$
10: **end while**
11: **return** D

AV の詳細なアルゴリズム

アルゴリズムは, Setup, v0 への代入, 1 回目の Predict, s0 への代入, Learn, v0 への代入, 2 回目の Predict という順で実行されるため, AV では 2 回目の Predict 実行後の $s1$ の依存関係を実行の逆順で解析していくことで検証を行う. AV の詳細なアルゴリズムを Algorithm 5 に示す. Algorithm 5 の入力 (Input) は検証対象のアルゴリズムであり, 出力 (Output) は検証対象のアルゴリズムが妥当な機械学習アルゴリズムであるかの真偽である. Algorithm 5 の各行の説明を以下に示す.

1-3 行目 PFV: Predict 関数の最後が s1 への代入命令であることを検証する. ここで, $\mathrm{Predict}_{\mathrm{last}}$ は Predict 関数の最後の命令を意味する.
4 行目 Predict 関数実行前の変数のうち, 予測ラベル $s1$ の算出に影響を与える変数をすべて求めて, $D_{\mathrm{Predict},s1}$ に格納する.
5-7 行目 PLV_{v0}: $s1$ が直前で代入された入力ベクトル $v0$ に依存していることを検証する.
8 行目 Predict 関数の直前に $v0$ への入力ベクトルの代入が存在するため, $D_{\mathrm{Predict},s1}$ から $v0$ を除外する.
9 行目 学習対象のパラメータが存在するかを表すフラグ learning_param_existence を false で

初期化する.

10-28 **行目** $D_{\mathrm{Predict},s1}$ 内に学習対象のパラメータを表す変数として適切なものが存在するかを検証する. 各ループでは, 候補 $c \in D_{\mathrm{Predict},s1}$ に対して, 学習対象のパラメータに関する検証 (LPV) を行う. 検証の結果, 学習対象のパラメータとして不適切であると判断された場合は, 当該候補の解析を終了し次の候補の解析に進む.

11 **行目** Learn 関数実行前の変数のうち, 学習対象のパラメータ候補である c の算出に影響を与える変数をすべて求めて, $D_{\mathrm{Learn},c}$ に格納する.

12-14 **行目** $\mathrm{LPV}_{\mathrm{prev}}$: 学習対象のパラメータの候補 c が更新される前の自分自身に依存していることを検証する.

15-17 **行目** LPV_{s0}: 学習対象のパラメータの候補 c が直前に代入された正解ラベル $s0$ に依存していることを検証する.

18-20 **行目** LPV_{s1}: 学習対象のパラメータの候補 c が直前に代入された予測ラベル $s1$ に依存していることを検証する.

21 **行目** Learn 関数の直前に $s0$ への正解ラベルの代入が存在するため, $D_{\mathrm{Learn},c}$ から $s0$ を除外する. また, 学習対象のパラメータが入力ベクトル $v0$ に依存しているかを検証する際は, $s1$ を介した間接的な依存は許さないため, $D_{\mathrm{Learn},c}$ から $s1$ も除外する.

22 **行目** Predict 関数実行前の変数のうち, 学習対象のパラメータ候補である c の算出に影響を与える変数をすべて求めて, $D_{\mathrm{Predict,Learn},c}$ に格納する.

23-25 **行目** LPV_{v0}: 学習対象のパラメータの候補 c が直前で代入された入力ベクトル $v0$ に依存していることを検証する.

26-27 **行目** LPV_{v0}: この行に到達することは, すべての学習対象のパラメータに関する検証に通ったことを意味するので, 学習対象のパラメータが存在するかを表すフラグ learning_param_existence を true にした上でループから抜ける.

19-31 **行目** $\mathrm{PLV}_{\mathrm{LP}}$: learning_param_existence が false である場合は, $s1$ が依存している変数の集合 $D_{\mathrm{Predict},s1}$ に学習対象のパラメータが含まれていないことを意味するので false を返却する.

Algorithm 5 AV のアルゴリズム

Input: $(\mathrm{Setup}, \mathrm{Predict}, \mathrm{Learn})$: 検証対象のアルゴリズム
Output: 検証対象のアルゴリズムが妥当であるかの真偽

1: **if** $\mathrm{output}(\mathrm{Predict}_{\mathrm{last}}) \neq s1$ **then**
2: **return** false
3: **end if**
4: $D_{\mathrm{Predict},s1} \leftarrow D(\mathrm{Predict}, \{s1\})$
5: **if** $v_0 \notin D_{\mathrm{Predict},s1}$ **then**
6: **return** false
7: **end if**
8: $D_{\mathrm{Predict},s1} \leftarrow D_{\mathrm{Predict},s1} \setminus \{v_0\}$
9: learning_param_existence $\leftarrow$ false
10: **for all** $c \in D_{\mathrm{Predict},s1}$ **do**
11: $D_{\mathrm{Learn},c} \leftarrow D(\mathrm{Learn}, \{c\})$
12: **if** $c \notin D_{\mathrm{Learn},c}$ **then**
13: continue
14: **end if**
15: **if** $s_0 \notin D_{\mathrm{Learn},c}$ **then**

```
        continue
    end if
    if s_1 ∉ D_{Learn,c} then
        continue
    end if
    D_{Learn,c} ← D_{Learn,c} \ {s_0, s1}
    D_{Predict,Learn,c} ← D(Predict, D_{Learn,c})
    if v_0 ∉ D_{Predict,Learn,c} then
        continue
    end if
    learning_param_existence ← true
    break
end for
if !learning_param_existence then
    return false
end if
return true
```

3.4 実験

本節では, 既存手法の RE-AutoML-Zero と提案手法の MGG-AutoML-Zero+AV の性能を比較するための線形回帰アルゴリズムの探索実験について述べる. 以下, 第 3.4.1 項では実験の目的, 第 3.4.2 項では比較手法, 第 3.4.3 項ではベンチマーク問題, 第 3.4.4 項では評価基準, 第 3.4.5 項では実験設定, 第 3.4.6 項では実験結果について述べる.

3.4.1 目的

本実験の目的は, 線形回帰アルゴリズムの探索問題において, 既存手法である Esteban らの手法 RE-AutoML-Zero に比べて, MGG による世代交代とアルゴリズムの検証を加えた提案手法 MGG-AutoML-Zero+AV が, 成功率および評価回数の観点で優れていることを確認することである.

3.4.2 比較手法

本実験では, 世代交代モデルとして RE を用いた Esteban らの手法 RE-AutoML-Zero と本論文の提案手法である MGG-AutoML-Zero+AV を比較する.

3.4.3 ベンチマーク問題

本実験では, 線形回帰のアルゴリズムが探索の対象であるため, $\mathcal{T}_{\text{search}}$ 内の各タスク $T^{(i)} (i \in \mathbb{N},\ 1 \leq i \leq 10)$ を以下の式で表される線形回帰問題とする.

$$T^{(i)} = \left\{ (\boldsymbol{x}_j^{(i)}, y_j^{(i)}) \mid \boldsymbol{x}_j^{(i)} \sim \boldsymbol{N}(0,1),\ y_j^{(i)} = \boldsymbol{w}_i \cdot \boldsymbol{x}_j^{(i)},\ \boldsymbol{w}_i \sim \boldsymbol{N}(0,1),\ j \in \mathbb{N},\ 1 \leq j \leq 100 \right\}$$

ここで, $\boldsymbol{v} \sim \boldsymbol{N}(0,1)$ は $\boldsymbol{v}$ の各要素が平均 0, 標準偏差 1 の正規乱数に従うことを意味する. 線型

回帰の問題の次元 $\dim(\boldsymbol{x}_j^{(i)}) = \dim(\boldsymbol{w}_i)$ は 4 に設定した．タスク $T^{(i)}$ のデータのうち, 100 件を学習データ $D_{\text{train}}^{(i)}$ とし, 残りの 100 件を検証用データ $D_{\text{valid}}^{(i)}$ とする．$\boldsymbol{w}_i$ はタスク $T^{(i)}$ ごとに設定されるため, 10 件の異なる勾配に従う線形回帰のデータセットがタスクとして与えられる点に注意されたい．

本実験では, $\mathcal{T}_{\text{search}}$ を使って探索した結果得られたアルゴリズム $\hat{a}$ が $\mathcal{T}_{\text{search}}$ にオーバーフィッティングしていないことを確認するために, $\mathcal{T}_{\text{search}}$ と独立に生成されたタスク集合 $\mathcal{T}_{\text{eval}}$ で $\hat{a}$ を評価する．$\mathcal{T}_{\text{eval}}$ の生成方法は $\mathcal{T}_{\text{search}}$ とほぼ同じだが，$1 \leq i \leq 100$，$1 \leq j \leq 1100$, $\left|D_{\text{train}}^{(i)}\right| = 1000$, $\left|D_{\text{valid}}^{(i)}\right| = 100$ となる点が異なる．

3.4.4 評価基準

本実験の性能評価指標として, タスク集合 $\mathcal{T}_{\text{search}}$ および $\mathcal{T}_{\text{eval}}$ を変えて, 10 試行分行ったときの探索成功率と十分な性能に到達するまでの平均の評価回数を用いる．ここで, 各試行における探索の成功とは, 評価回数が 10,000,000 回に到達する前に, 集団内で最も $\mathcal{T}_{\text{search}}$ に対する評価値が高いアルゴリズム $\hat{a}$ が, $\mathcal{T}_{\text{eval}}$ に対して評価値 0.999 以上になることをいう．また, 十分な性能とは成功試行において集団内の $\mathcal{T}_{\text{search}}$ に対する最高評価値が 0.999 を超えることをいう．

3.4.5 実験設定

RE-AutoML-Zero, MGG-AutoML-Zero+AV いずれの手法でも, 集団サイズ N_{pop} は 1200 とし, 変数の個数はスカラーが 4, ベクトルを 3, 行列は 1 とした．また, アルゴリズムを構成する各関数に含まれる命令の制約もすべての手法で統一した．Setup は命令の個数が 5 つ固定で命令の種類は OP56, OP57 のみ, Predict は命令の個数が 1 つ固定で命令の種類は OP27 のみ, Learn は命令の個数が 4 つ固定で命令の種類は OP1, OP2, OP3, OP18, OP23 のみとした．命令の ID と内容の対応関係は, Table.A.1 を参照されたい．変数の最大値および Setup, Predict, Learn に対する制約は, Esteban らの論文 [11] の Section 4.1 の線型回帰に関する実験と同じで, 線形回帰のアルゴリズムを構成するのに十分な探索範囲に限定した．突然変異手法も Esteban らの論文 [11] の Section 4.1 の線型回帰の実験と同様に, 第 2.4.4 項の (1) を除外した．RE-AutoML-Zero では突然変異確率 p_{mutate} を 0.9, トーナメントサイズ K を 10 に設定し, MGG-AutoML-Zero+AV では子個体生成数 N_{c} を 800 とした．

3.4.6 実験結果

Table.3.3 に RE-AutoML-Zero と MGG-AutoML-Zero+AV を比較した実験の結果を示す．実験結果から分かるように, RE-AutoML-Zero は成功率が 5 割であることに対し, MGG-AutoML-Zero+AV はすべての試行に対して成功している． また, 平均評価回数も MGG-AutoML-Zero+AV は RE-AutoML-Zero の 1/100 未満になっている．

3.5 考察

本節では, 第 3.4 節で行った提案手法と既存手法の比較実験に関する考察および提案手法の分析を行う．第 3.5.1 項では, 第 3.4 節で行った実験における集団内の最良個体の評価値の推移を示して, 実験結果の考察を行う．第 3.5.2 項では, MGG-AutoML-Zero と RE-AutoML-Zero を比較し, MGG の有効性を確認した上で, MGG が集団の多様性維持にどの程度影響を与えているのかを考察する．第 3.5.3 項では, MGG-AutoML-Zero と MGG-AutoML-Zero+AV を比較し, AV

Table.3.3　RE-AutoML-Zero および MGG-AutoML-Zero+AV で線形回帰アルゴリズムを探索した結果

No.	RE-AutoML-Zero		MGG-AutoML-Zero+AV	
	結果	評価回数	結果	評価回数
1	失敗	-	成功	34800
2	失敗	-	成功	12400
3	失敗	-	成功	46000
4	成功	51,600	成功	18000
5	成功	6,553,200	成功	23600
6	成功	606,000	成功	40400
7	失敗	-	成功	1200
8	成功	4,738,800	成功	40400
9	失敗	-	成功	6800
10	成功	5,948,400	成功	57200
成功試行数	5/10		10/10	
平均の評価回数	3,579,600		28,080	

の有効性を確認した上で, AV によってどの程度探索空間が削減されたかを考察する．また, AV 内で導入されている各検証がすべて有効であることを確認するために, MGG-AutoML-Zero+AV から一部の検証を外した手法と MGG-AutoML-Zero+AV を比較して分析を行う．

3.5.1　実験結果について

第 3.4 節で述べたように, MGG-AutoML-Zero+AV は, RE-AutoML-Zero が半分の試行でしか成功しなかった線形回帰タスクの実験において, すべての試行を成功させ, 評価値も 1/100 未満にすることが出来た．この結果を詳細に分析するために, 第 3.4 節の実験における集団内の最良個体の評価値の推移を Fig.3.3 と Fig.3.4 を示す．横軸は評価回数であり, 縦軸は集団内の最良個体の評価値である．Fig.3.4 は, Fig.3.3 から MGG-AutoML-Zero+AV のみを取り出して拡大したグラフである．集団内の最良個体の評価値の推移から読み取れるように, MGG-AutoML-Zero+AV は, どの試行でも評価回数が少ない段階で安定して評価値の改善が進んでいる．一方, RE-AutoML-Zero は特定の評価値付近で長期間改善されていない状況, すなわち局所解に陥っている状況が発生していることが分かる．

3.5.2　MGG の有効性の確認と分析

本節では, 第 3.4 節と同じ実験設定で MGG-AutoML-Zero と RE-AutoML-Zero の比較実験を行い, MGG の有効性を確認した上で, MGG よって集団の多様性がどの程度維持されるようになったのかを分析する．以下, 第 3.5.2 節では MGG-AutoML-Zero と RE-AutoML-Zero の比較実験による MGG の有効性確認, 第 3.5.2 節では MGG による多様性維持の分析を行う．

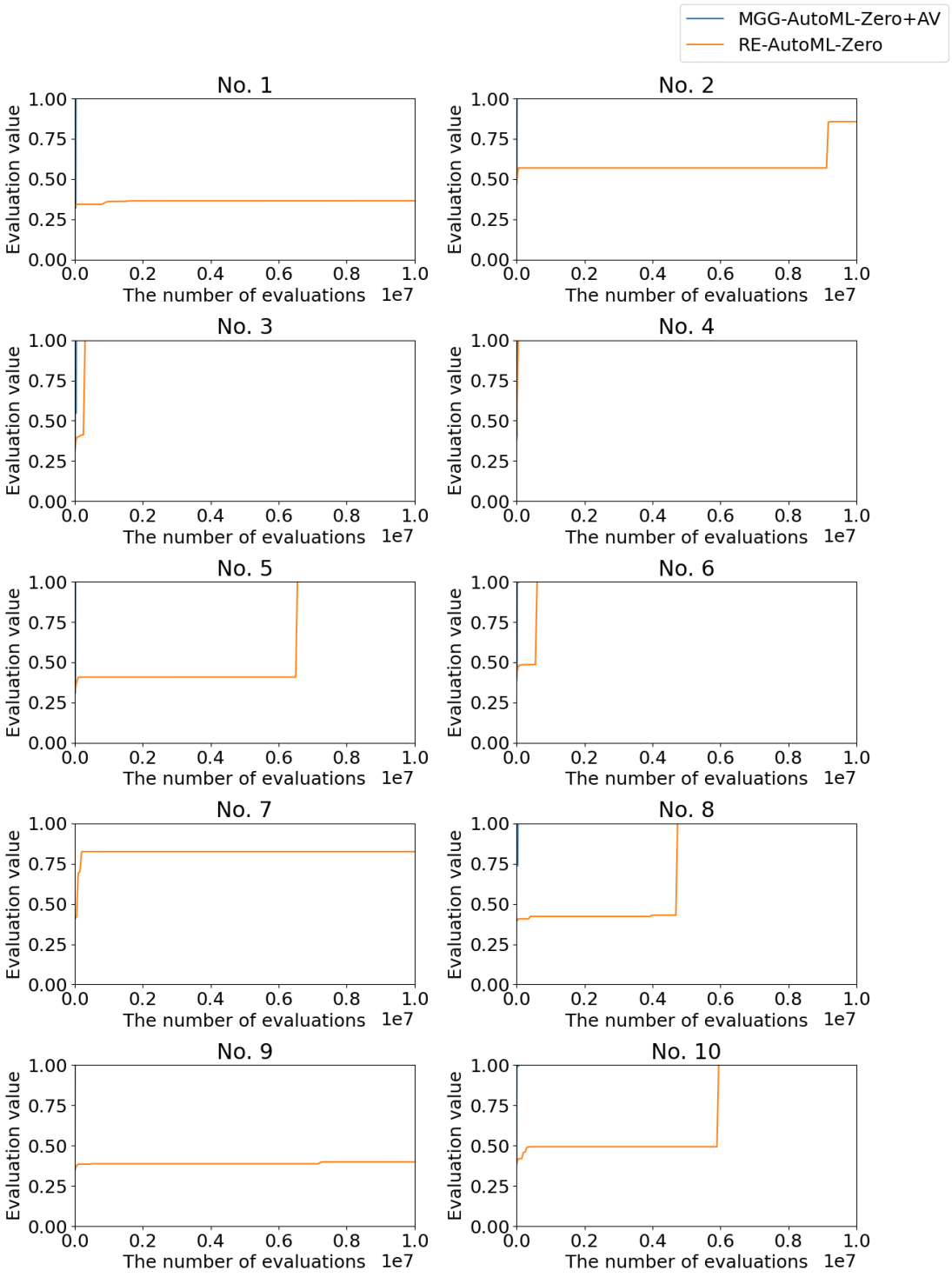

Fig.3.3　第 3.4 節の実験における MGG-AutoML-Zero+AV と AutoML-Zero の集団内の最良個体の評価値の推移. 横軸は評価回数であり, 縦軸は集団内の最良個体の評価値である.

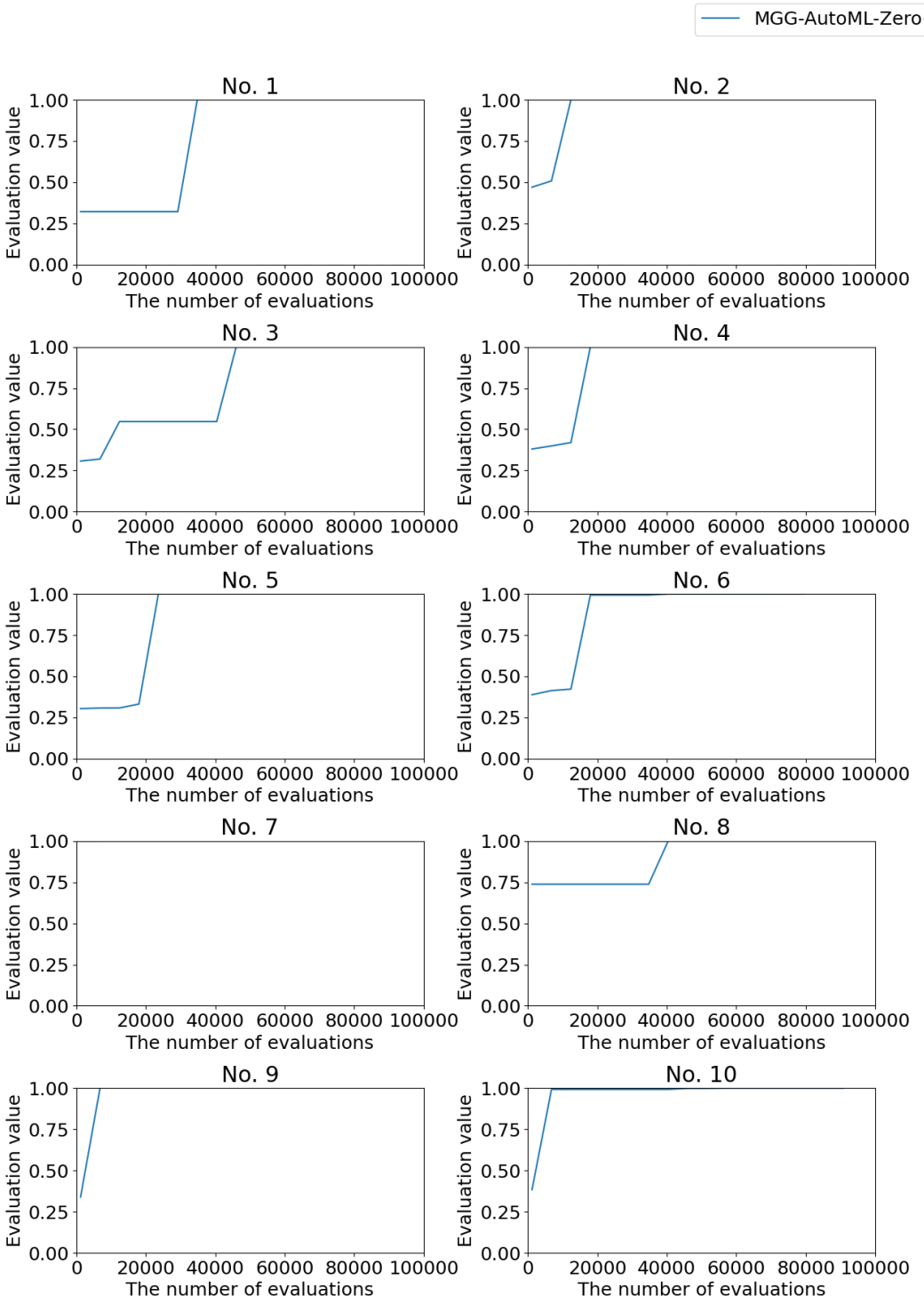

Fig.3.4　第 3.4 節の実験における MGG-AutoML-Zero+AV の集団内の最良個体の評価値の推移．横軸は評価回数であり，縦軸は集団内の最良個体の評価値である．No. 7 の試行は初期集団で評価値が 0.999 を超える個体が生成されているため，グラフが 1.00 の横軸と重なっていることに注意されたい．

Table.3.4　第 3.4 節と同じ実験設定で MGG-AutoML-Zero と RE-AutoML-Zero を比較した結果.

No.	RE-AutoML-Zero		MGG-AutoML-Zero	
	結果	評価回数	結果	評価回数
1	失敗	-	成功	2,874,000
2	失敗	-	成功	4,486,800
3	失敗	-	成功	3,630,000
4	成功	51,600	成功	1,412,400
5	成功	6,553,200	成功	2,017,200
6	成功	606,000	成功	7,006,800
7	失敗	-	成功	2,571,600
8	成功	4,738,800	成功	5,192,400
9	失敗	-	失敗	-
10	成功	5,948,400	成功	908,400
成功率	5/10		9/10	
評価回数の平均	3,579,600		3,344,400	

MGG の有効性の確認

本項では, 第 3.4 節と同じ実験設定で MGG-AutoML-Zero と RE-AutoML-Zero の比較実験を行うことで, MGG の有効性を確認する．本実験では, 初期集団の段階で多様性に差がでないよう初期集団を MGG-AutoML-Zero と RE-AutoML-Zero で揃えた．MGG-AutoML-Zero と RE-AutoML-Zero の比較実験結果を, Table.3.4 に示す．Table.3.4 の実験結果から, MGG-AutoML-Zero は RE-AutoML-Zero と比べて探索の成功率は 40% 改善し, 評価回数は 200,00 回程度少なくなることが分かった．したがって, 本研究で提案した MGG による世代交代は, アルゴリズムの探索に対して有効に作用することが確認された.

MGG による多様性維持の分析

本項では, 第 3.5.2 節で行った実験における集団内の最良個体の評価値の推移を確認することで, MGG の方が集団の多様性を維持できる世代交代モデルであることを確認する．RE-AutoML-Zero および MGG-AutoML-Zero それぞれの集団内の最良個体の評価値の推移を Fig.3.5 に示す. Fig.3.5 から MGG-AutoML-Zero は RE-AutoML-Zero に比べて, 段階的に集団内の最良個体の評価値が上昇していて, 局所解に陥りにくいことが分かる．これは MGG-AutoML-Zero の方が, 集団の中に評価値を改善できる見込みがある形質を持った個体が, 集団内に多く含まれていることを意味する．つまり, MGG-AutoML-Zero は RE-AutoML-Zero に比べて, 集団の多様性を維持しやすいと考えられる.

3.5.3　AV の有効性の確認と分析

本節では, 第 3.4 節と同じ実験設定で MGG-AutoML-Zero+AV と MGG-AutoML-Zero の比較実験を行い, AV の有効性を確認した上で, AV によってどの程度探索空間が削減されたのかを分析する．また, AV で行う各検証が有効であるかどうかを調べるために, MGG-AutoML-

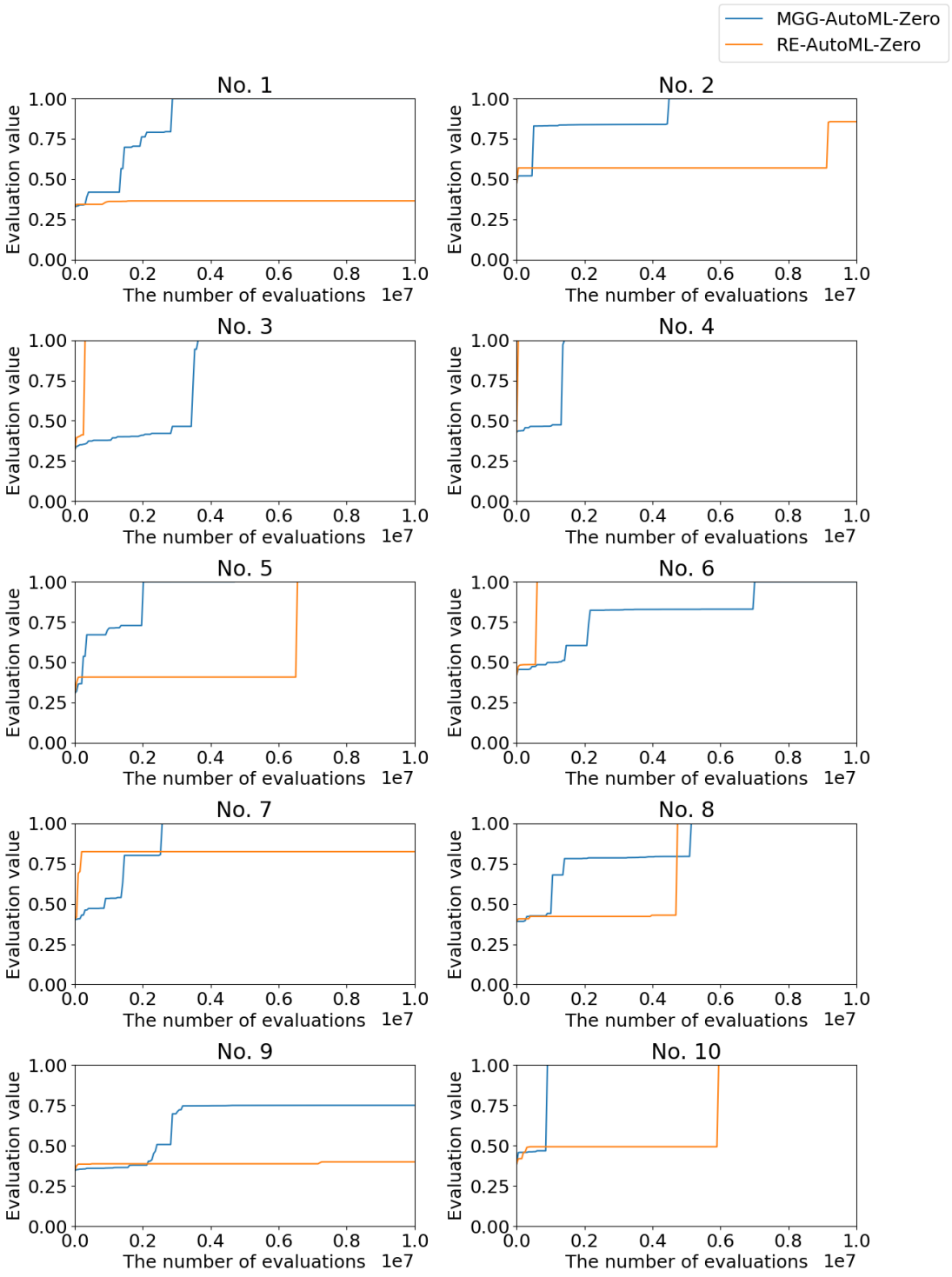

Fig.3.5　第 3.5.2 節で行った RE-AutoML-Zero および MGG-AutoML-Zero の比較実験における最良評価値の推移. 横軸は評価回数であり, 縦軸は集団内の最良個体の評価値である.

Table.3.5　RE-AutoML-Zero, MGG-AutoML-Zero, MGG-AutoML-Zero+AV で線形回帰アルゴリズムを探索した結果

No.	MGG-AutoML-Zero		MGG-AutoML-Zero+AV	
	結果	評価回数	結果	評価回数
1	成功	2,874,000	成功	34800
2	成功	4,486,800	成功	12400
3	成功	3,630,000	成功	46000
4	成功	1,412,400	成功	18000
5	成功	2,017,200	成功	23600
6	成功	7,006,800	成功	40400
7	成功	2,571,600	成功	1200
8	成功	5,192,400	成功	40400
9	失敗	-	成功	6800
10	成功	908,400	成功	57200
成功率	9/10		10/10	
評価回数の平均	3,344,400		28,080	

Zero+AV と MGG-AutoML-Zero+AV から一部の検証を外した手法を比較する. 以下, 第 3.5.3 節では MGG-AutoML-Zero+AV と MGG-AutoML-Zero の比較実験による AV の有効性確認, 第 3.5.3 節では AV による探索空間の削減に関する分析, 第 3.5.3 節では AV における各検証の有効性の確認を行う.

AV の有効性の確認

本項では, 第 3.4 節と同じ実験設定で MGG-AutoML-Zero+AV と MGG-AutoML-Zero の比較実験を行うことで, AV の有効性を確認する. MGG-AutoML-Zero+AV と MGG-AutoML-Zero の比較実験結果を, Table.3.5 に示す. Table.3.5 の実験結果から, MGG-AutoML-Zero+AV は MGG-AutoML-Zero が失敗していた No. 9 の試行に対しても成功し, 評価回数は 1/100 未満になることが分かった. したがって, 本研究で提案したアルゴリズムの検証 AV は, アルゴリズムの探索に対して有効に作用することが確認された.

AV の分析

本項では, AV によってどの程度探索空間を削減できたのかを定量化する. 第 3.5.3 節で行った MGG-AutoML-Zero+AV の実験において, アルゴリズムの検証で妥当な機械学習アルゴリズムと判断された個体の割合を, 初期集団生成時と子個体生成時に分けて, Table.3.6 に示す. Table.3.6 から完全にランダムに個体を生成する初期集団生成時には, 0.0138% アルゴリズムのみが実行可能解 (妥当な機械学習アルゴリズム) であり, その他の 99.9862% は実行不可能解であった. このことから, アルゴリズムの検証を導入しない既存手法では, 探索空間の 99.9862% が冗長な探索空間になっていたことが分かる. また, 妥当な機械学習アルゴリズムの突然変異によって個体を生成する子個体生成では, 妥当な機械学習アルゴリズム生成される割合は 0.0264% であった. すなわち, RE-AutoML-Zero の突然変異オペレータは, 実行可能解を不可能解にしてしまう確率が 99.97% を超える破壊的なオペレータであることが分かった.

Table.3.6　第 3.5.3 節で行った MGG-AutoML-Zero+AV の実験において, アルゴリズムの検証で実行可能解 (妥当な機械学習アルゴリズム) と判断された個体の割合. 初期集団生成時と子個体生成時に分けて示している.

No	初期集団生成時	子個体生成時
1	0.0140%	0.0260%
2	0.0137%	0.0269%
3	0.0133%	0.0262%
4	0.0142%	0.0261%
5	0.0138%	0.0258%
6	0.0143%	0.0261%
7	0.0131%	0.0276%
8	0.0139%	0.0266%
9	0.0138%	0.0263%
10	0.0137%	0.0262%
平均	0.0138%	0.0264%

AV で行う各検証の有効性の確認

本節では, AV の各検証の有効性を確認するために, AV から特定の検証 X を除外したアルゴリズム検証 AV $-$ X を用いて第 3.4 節と同じ実験を行う. つまり, 探索に用いる手法は, MGG-AutoML-Zero+(AV $-$ X) となる. X には, PFV, PLV_{v0}, LPV_{s0} 等の検証名を代入する. 検証名と検証内容の対応関係の詳細は第 3.3.2 節を参照されたい.

Table.3.7 にアルゴリズムの検証法ごとに, 10 試行分の実験を行った結果を示す. 探索の成功率は 10 試行の中で成功した探索の割合, 平均の評価回数は成功試行の探索が要した評価回数の平均, 平均の実行可能解の割合は, 初期集団生成時と子個体生成時それぞれで実行可能解が生成された割合の平均を意味する. Table.3.7 より, AV を検証法として導入したときが最も平均の評価回数が小さいことが分かる. ただし, 今回の実験においては Predict 関数の命令数が 1 で固定されているため, PFV は PLV_{v0} や PLV_{LP} の検証に包含されてしまうため, AV $-$ PFV と AV が同じ結果になった. しかし, PFV は Predict に含まれる関数が複数になった場合, 他の検証をするために必要な検証であるため, 欠かすことは出来ない. 以上のことから, AV に含まれるすべての検証が探索に対して有効であったことが従う.

3.6　おわりに

本章では, Esteban らが提案した手法である RE-AutoML-Zero の問題点を述べ, それらの問題を解決する手法 MGG-AutoML-Zero+AV を提案した. そして, MGG-AutoML-Zero+AV と RE-AutoML-Zero の比較実験および考察を通じて, 提案手法の有効性を確認した. 以下では, 本章の内容を簡単に取りまとめる.

Table.3.7　検証法ごとに第 3.4 節の実験を行った結果. 探索の成功率は 10 試行の中で成功した探索の割合, 平均の評価回数は成功試行の探索が要した評価回数の平均, 平均の実行可能解の割合は, 初期集団生成時と子個体生成時それぞれで実行可能解が生成された割合の平均を意味する.

検証手法	探索の成功率	平均の評価回数	平均の実行可能解の生成割合	
			初期集団生成時	子個体生成時
なし	9/10	3,344,400	100.0000%	100.0000%
AV	10/10	28,080	0.0138%	0.0264%
AV − PFV	10/10	28,080	0.0138%	0.0264%
AV − PLV_{v0}	10/10	62,800	0.1055%	0.2174%
AV − $\mathrm{PLV}_{\mathrm{LP}}$	6/10	2,580,000	55.6808%	80.6304%
AV − $\mathrm{LPV}_{\mathrm{prev}}$	10/10	138400	0.4088%	0.8165%
AV − LPV_{s0}	10/10	162480	0.3710%	0.7364%
AV − LPV_{s1}	10/10	197760	0.4205%	0.8504%
AV − LPV_{v0}	10/10	151840	0.3030%	0.6081%

3.6.1　既存手法の問題点

本章では, 第 2 章で AutoML-Zero の問題設定を説明した上で, Esteban らが提案した手法である RE-AutoML-Zero に関する問題点を 2 つ指摘した. 1 つめは, RE-AutoML-Zero で用いられている世代交代モデル RE は多様性を失いやすい点である. RE は, 複製選択で選択圧が強く掛かったり, 生存選択で無条件で淘汰する個体が淘汰され, 淘汰される個体とは関係ない個体が生存したりすることがあり, それが集団の多様性の低下に繋がっている可能性を指摘した. 2 つめとしては, 妥当ではない機械学習アルゴリズムが探索対象となっており, 探索空間が冗長になっている問題点を指摘した. RE-AutoML-Zero では, 学習対象のパラメータや入力ベクトルが予測のラベルを求めるのに使われていなくても, 探索対象として扱い評価を行っていた. しかし, このような妥当ではないアルゴリズムは, いかなる機械学習のタスクについても, 除外されるべきであるため, 探索対象に含める必要がないことを論じた.

3.6.2　既存手法の問題点に対処した手法の提案

本章の第 3.3 節では, 既存手法の問題点に対処した手法を提案した. はじめに, 集団の多様性が失われやすい世代交代である RE を, 佐藤らが提案した MGG に置き換えることを提案した. MGG は複製選択時の選択圧を無くした上で, 生存選択時には次の世代に追加される個体として, 淘汰された個体自身または家族が選択される世代交代モデルであり, 既存手法の世代交代モデルに関する問題に対処できる手法である. また, 妥当ではない機械学習アルゴリズムが探索対象となる問題については, 妥当なアルゴリズムであることを検証して, 妥当ではないアルゴリズムを実行不可能解として扱う手法を提案した.

3.6.3 提案手法の性能評価と考察

本章では, 第 3.4 節で提案手法が既存手法よりも線型回帰タスクの問題において優れていることを確認した上で, 第 3.5 節で既存手法の問題点に対して対処できていることを分析によって明らかにした. 第 3.4 節の実験結果では, 既存手法が 50% しか成功しない線型回帰タスクに対して, 提案手法は 100% 成功した上で評価回数も既存手法の 1/100 未満になることを確認した.

第 3.5.2 項では, 世代交代モデルとして MGG を導入した MGG-AutoML-Zero と既存手法の RE-AutoML-Zero の 2 つを比較をすることで MGG の有効性を確認した. MGG-AutoML-Zero は RE-AutoML-Zero に比べて成功率が 40% 高く, 探索に必要な評価回数も 200,000 回以上少ないことが分かった. また, 集団内の最良個体の評価値の推移を比較することで, 集団の多様性が RE に比べて MGG の方が維持できていることを確認した.

第 3.5.3 項では, アルゴリズムの検証に関する有効性を MGG-AutoML-Zero と MGG-AutoML-Zero+AV を比較することで確認した. MGG-AutoML-Zero+AV は, MGG-AutoML-Zero で解けなかった試行が解けるようになり, 評価回数も 1/100 未満になることが分かった. 加えて, アルゴリズムの検証で実行不可能解となるアルゴリズムの割合を調べることで, 探索空間をどの程度削減できているのかを分析した. 分析の結果, 既存手法の探索空間の 99.98% 以上が削減できていることが分かった.

付録 A

命令セット

本研究のアルゴリズムを構成するための命令セットを Table.A.1 に示す．この命令セットは Esteban らの論文 [11] に使われている命令セットと同様である．

Table.A.1: アルゴリズムの各関数 Setup, Predict, Learn を構成する命令の一覧．Esteban らの論文 [11] に示されている命令セットと同様である．

ID	コード例	入力		出力		説明
		変数/型	定数	変数/型	添字	
OP0	no_op	-	-	-	-	-
OP1	$s2 = s3 + s0$	a, b/scalars	-	c/scalar	-	$s_c = s_a + s_b$
OP2	$s4 = s0 - s1$	a, b/scalars	-	c/scalar	-	$s_c = s_a - s_b$
OP3	$s8 = s5 * s5$	a, b/scalars	-	c/scalar	-	$s_c = s_a s_b$
OP4	$s7 = s5/s2$	a, b/scalars	-	c/scalar	-	$s_c = s_a / s_b$
OP5	$s8 = \mathrm{abs}(s0)$	a/scalar	-	b/scalar	-	$s_b = \lvert s_a \rvert$
OP6	$s4 = 1/s8$	a/scalar	-	b/scalar	-	$s_b = 1/s_a$
OP7	$s5 = \sin(s4)$	a/scalar	-	b/scalar	-	$s_b = \sin(s_a)$
OP8	$s1 = \cos(s4)$	a/scalar	-	b/scalar	-	$s_b = \cos(s_a)$
OP9	$s0 = \tan(s4)$	a/scalar	-	b/scalar	-	$s_b = \tan(s_a)$
OP10	$s0 = \arcsin(s4)$	a/scalar	-	b/scalar	-	$s_b = \arcsin(s_a)$
OP11	$s2 = \arccos(s0)$	a/scalar	-	b/scalar	-	$s_b = \arccos(s_a)$
OP12	$s4 = \arctan(s0)$	a/scalar	-	b/scalar	-	$s_b = \arctan(s_a)$
OP13	$s1 = \exp(s2)$	a/scalar	-	b/scalar	-	$s_b = e^{s_a}$
OP14	$s0 = \log(s3)$	a/scalar	-	b/scalar	-	$s_b = \log s_a$
OP15	$s3 = \mathrm{heaviside}(s0)$	a/scalar	-	b/scalar	-	$s_b = \mathbb{1}_{\mathbb{R}+}(s_a)$
OP16	$v2 = \mathrm{heaviside}(v2)$	a/vector	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \mathbb{1}_{\mathbb{R}+}(\boldsymbol{v}_a^{(i)})$
OP17	$m7 = \mathrm{heaviside}(m3)$	a/matrix	-	b/matrix	-	$\forall i, jM_b^{(i,j)} = \mathbb{1}_{\mathbb{R}+}(M_a^{(i,j)})$
OP18	$v1 = s7 * v1$	a, b/sc, vec	-	c/vector	-	$\boldsymbol{v}_c = s_a \boldsymbol{v}_b$
OP19	$v1 = \mathrm{bcast}(s3)$	a/scalar	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = s_a$
OP20	$v5 = 1/v7$	a/vector	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = 1/s_a^{(i)}$
OP21	$s0 = \mathrm{norm}(v3)$	a/scalar	-	b/vector	-	$s_b = \lVert \boldsymbol{v}_a \rVert$
OP22	$v3 = \mathrm{abs}(v3)$	a/vector	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\lvert \boldsymbol{v}_a^{(i)} \right\rvert$
OP23	$v5 = v0 + v9$	a, b/vectors	-	c/scalar	-	$\boldsymbol{v}_c = \boldsymbol{v}_a + \boldsymbol{v}_b$
OP24	$v1 = v0 - v9$	a, b/vectors	-	c/scalar	-	$\boldsymbol{v}_c = \boldsymbol{v}_a - \boldsymbol{v}_b$
OP25	$v8 = v0 * v9$	a, b/vectors	-	c/scalar	-	$\forall i, \boldsymbol{v}_c^{(i)} = \boldsymbol{v}_a^{(i)} \boldsymbol{v}_b^{(i)}$
OP26	$v9 = v8/v2$	a, b/vectors	-	c/scalar	-	$\forall i, \boldsymbol{v}_c^{(i)} = \boldsymbol{v}_a^{(i)} / \boldsymbol{v}_b^{(i)}$

命令 ID	コード例	入力		出力		説明
		変数/型	定数	変数/型	添字	
OP27	$s6 = \mathrm{dot}(v1, v5)$	a, b/vectors	-	c/scalar	-	$s_c = \boldsymbol{v}_a^T \boldsymbol{v}_b$
OP28	$m1 = \mathrm{outer}(v6, v5)$	a, b/vectors	-	c/matrix	-	$M_c = \boldsymbol{v}_a \boldsymbol{v}_b^T$
OP29	$m1 = a4 * m2$	a, b/sc/mat	-	c/matrix	-	$M_c = s_a M_b$
OP30	$m3 = 1/m0$	a/matrix	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = 1/M_a^{(i,j)}$
OP31	$v6 = \mathrm{dot}(m1, v0)$	a, b/mat/vec	-	c/vector	-	$\boldsymbol{v}_c = M_a \boldsymbol{v}_b$
OP32	$m2 = \mathrm{bcast}(v0, \mathrm{axis} = 0)$	a/vector	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = \boldsymbol{v}_a^{(i)}$
OP33	$m2 = \mathrm{bcast}(v0, \mathrm{axis} = 1)$	a/vector	-	b/matrix	-	$\forall i, j, M_b^{(j,i)} = \boldsymbol{v}_a^{(i)}$
OP34	$s2 = \mathrm{norm}(m1)$	a/matrix	-	b/scalar	-	$s_b = \lVert \boldsymbol{v}_a \rVert$
OP35	$v4 = \mathrm{norm}(m7, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\lVert M_a^{(i,\cdot)} \right\rVert$
OP36	$v4 = \mathrm{norm}(m7, \mathrm{axis} = 1)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\lVert M_a^{(\cdot,i)} \right\rVert$
OP37	$m9 = \mathrm{transpose}(m3)$	a/matrix	-	b/matrix	-	$M_b = M_a^T$
OP38	$m1 = \mathrm{abs}(m8)$	a/matrix	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = \left\lvert M_a^{(i,j)} \right\rvert$
OP39	$m2 = m2 + m0$	a, b/matrixes	-	c/matrix	-	$M_c = M_a + M_b$
OP40	$m2 = m3 - m1$	a, b/matrixes	-	c/matrix	-	$M_c = M_a - M_b$
OP41	$m3 = m2 * m3$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = M_a^{(i,j)} M_b^{(i,j)}$
OP42	$m4 = m2/m4$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = M_a^{(i,j)} / M_b^{(i,j)}$
OP43	$m5 = \mathrm{matmul}(m5, m7)$	a, b/matrixes	-	c/matrix	-	$M_c = M_a M_b$
OP44	$s1 = \mathrm{minimun}(s2, s3)$	a, b/scalars	-	c/scalar	-	$s_c = \min(s_a, s_b)$
OP45	$v4 = \mathrm{minimun}(v3, v9)$	a, b/vectors	-	c/vector	-	$\forall i, \boldsymbol{v}_c^{(i)} = \min\left(\boldsymbol{v}_a^{(i)}, \boldsymbol{v}_b^{(i)}\right)$
OP46	$m5 = \mathrm{minimun}(m5, m7)$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = \min\left(M_a^{(i,j)}, M_b^{(i,j)}\right)$
OP47	$s8 = \mathrm{maximum}(s3, s0)$	a, b/scalars	-	c/scalar	-	$s_c = \max(s_a, s_b)$
OP48	$v7 = \mathrm{maximum}(v3, v6)$	a, b/vectors	-	c/vector	-	$\forall i, \boldsymbol{v}_c^{(i)} = \max\left(\boldsymbol{v}_a^{(i)}, \boldsymbol{v}_b^{(i)}\right)$
OP49	$m7 = \mathrm{maximum}(m1, m0)$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = \max\left(M_a^{(i,j)}, M_b^{(i,j)}\right)$
OP50	$s2 = \mathrm{mean}(v2)$	a/vector	-	b/scalar	-	$s_b = \mathrm{mean}(\boldsymbol{v}_a)$
OP51	$s2 = \mathrm{mean}(m8)$	a/matrix	-	b/scalar	-	$s_b = \mathrm{mean}(M_a)$
OP52	$v1 = \mathrm{mean}(m2, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \mathrm{mean}\left(M_a^{(i,\cdot)}\right)$
OP53	$v3 = \mathrm{stdev}(m2, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \mathrm{stdev}\left(M_a^{(i,\cdot)}\right)$
OP54	$s3 = \mathrm{stdev}(v3)$	a/vector	-	b/scalar	-	$s_b = \mathrm{stdev}(\boldsymbol{v}_a)$
OP55	$s4 = \mathrm{stdev}(m0)$	a/matrix	-	b/scalar	-	$s_b = \mathrm{stdev}(M_a)$
OP56	s2 = 0.7	-	γ	a/scalar	-	$s_a = \gamma$
OP57	v3[5] = -2.4	-	γ	a/vector	i	$\boldsymbol{v}_a^{(i)} = \gamma$
OP58	m2[5,1] = -0.03	-	γ	a/matrix	i, j	$M^{(i,j)} = \gamma$
OP59	$s4 = \mathrm{uniform}(-1, 1)$	-	α, β	a/scalar	-	$s_a = \mathcal{U}(\alpha, \beta)$
OP60	$v1 = \mathrm{uniform}(0.4, 0.8)$	-	α, β	a/vector	-	$\forall i, \boldsymbol{v}_a^{(i)} = \mathcal{U}(\alpha, \beta)$
OP61	$m0 = \mathrm{uniform}(-0.5, 0.6)$	-	α, β	a/vector	-	$\forall i, j, M_a^{(i,j)} = \mathcal{U}(\alpha, \beta)$
OP62	$s4 = \mathrm{gaussian}(0.1, 0.7)$	-	μ, σ	a/scalar	-	$s_a = \mathcal{N}(\mu, \sigma)$
OP63	$v8 = \mathrm{gaussian}(0.4, 1)$	-	μ, σ	a/vector	-	$\forall i, \boldsymbol{v}_a^{(i)} = \mathcal{N}(\mu, \sigma)$
OP64	$m2 = \mathrm{gaussian}(-2, 1.3)$	-	μ, σ	a/vector	-	$\forall i, j, M_a^{(i,j)} = \mathcal{N}(\mu, \sigma)$

参考文献

[1] Ferran Alet, Martin F. Schneider, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Meta-learning curiosity algorithms. In International Conference on Learning Representations, 2020.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. Advances in neural information processing systems, Vol. 29, pp. 3981–3989, 2016.

[3] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 2019.

[4] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. The Journal of Machine Learning Research, Vol. 20, No. 1, pp. 1997–2017, 2019.

[5] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Advances in neural information processing systems, Vol. 2, pp. 524–532, 1989.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning, pp. 1126–1135. PMLR, 2017.

[7] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5, pp. 507–523. Springer, 2011.

[8] Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. NAS-bench-suite: NAS evaluation is (now) surprisingly easy. In International Conference on Learning Representations, 2022.

[9] Renato Negrinho, Matthew Gormley, Geoffrey J Gordon, Darshan Patil, Nghia Le, and Daniel Ferreira. Towards modular and programmable architecture search. Advances in neural information processing systems, Vol. 32, pp. 524–532, 2019.

[10] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In Proceedings of the aaai conference on artificial intelligence, Vol. 33, pp. 4780–4789, 2019.

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

[12] Hiroshi Sato. A new generation alternation model of genetic algorithms and its assessment. Transactions of the Japanese Society for Artificial Intelligence, Vol. 12, No. 2, pp.

82–91, 1997.

[13] David So, Quoc Le, and Chen Liang. The evolved transformer. In International conference on machine learning, pp. 5877–5886. PMLR, 2019.

[14] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2820–2828, 2019.

[15] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. 2017.

[16] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710, 2018.