集団の多様性維持と探索空間の削減に着目した AutoML-Zero の提案

三嶋隆史

2023 年 04 月 30 日

概要

前回の研究会

- MGG-AutoML-Zero+VAT の提案
- MGG-AutoML-Zero+VAT の性能を線形回帰とアフィン回帰問題、非線形回帰問題で確認

今回の研究会

- MGG-AutoML-Zero+VAT の非線形回帰問題に対する失敗試行を考察
- MGG-AutoML-Zero+VAT の非線形回帰問題で発見されたアルゴリズムの解析
- 考察した結果、分かった MGG-AutoML-Zero+VAT の問題点を整理
- MGG-AutoML-Zero+VAT の問題点に対処した手法の提案（現時点ではまだアイディアベース）

注意事項

今回からは、卒業論文のときの手法 MGG-AutoML-Zero+AV や関連する手法の説明は省略し, Esteban らが提案した RE-AutoML-Zero と現時点での最新手法のみを説明することにする.

また, 今回の研究会では研究室に新しく入った方々もいるので, 基本的な考え方に重点を置いて説明する.

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.

これまでの AutoML に関する研究の多くは, 計算コストを抑えるために人間のデザインに大きく依存した制約付きの空間を探索している. 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].

こうした探索空間を限定する AutoML には, 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, 集団の多様性の低下の問題, 探索空間の冗長性の問題, 突然変異のランダム性が非常に高い問題に着目した. 本論文の目的は, これらの Esteban らの AutoML-Zero の問題点に対処した手法を提案し, 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.

1.3 本論文の構成

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.

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

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 $\mathcal{T}_{\text{eval}}$ は独立に構成する.

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. 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

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

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

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** $e = 0, 1, \cdots, N_{\text{epochs}}$ **do**
4: **for all** $(\boldsymbol{x}^{(i)}_j, y^{(i)}_j) \in D^{(i)}_{\text{train}}$ **do**
5: $v0 \leftarrow \boldsymbol{x}^{(i)}_j$
6: Predict()
7: $s1 \leftarrow \text{Normalize}(s1)$
8: $s0 \leftarrow y^{(i)}_j$
9: Learn()
10: **end for**
11: **end for**
12: $l_{\text{sum}} = 0.0$
13: **for all** $(\boldsymbol{x}^{(i)}_j, y^{(i)}_j) \in D^{(i)}_{\text{valid}}$ **do**
14: $v0 \leftarrow \boldsymbol{x}^{(i)}_j$
15: Predict()
16: $s1 \leftarrow \text{Normalize}(s1)$
17: $l_{\text{sum}} \leftarrow l_{\text{sum}} + \text{Loss}(y, s1)$
18: **end for**
19: $l_{\text{mean}} \leftarrow l_{\text{sum}} / |D_{\text{valid}}|$
20: fitness = $\text{Rescale}(l_{\text{mean}})$
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．

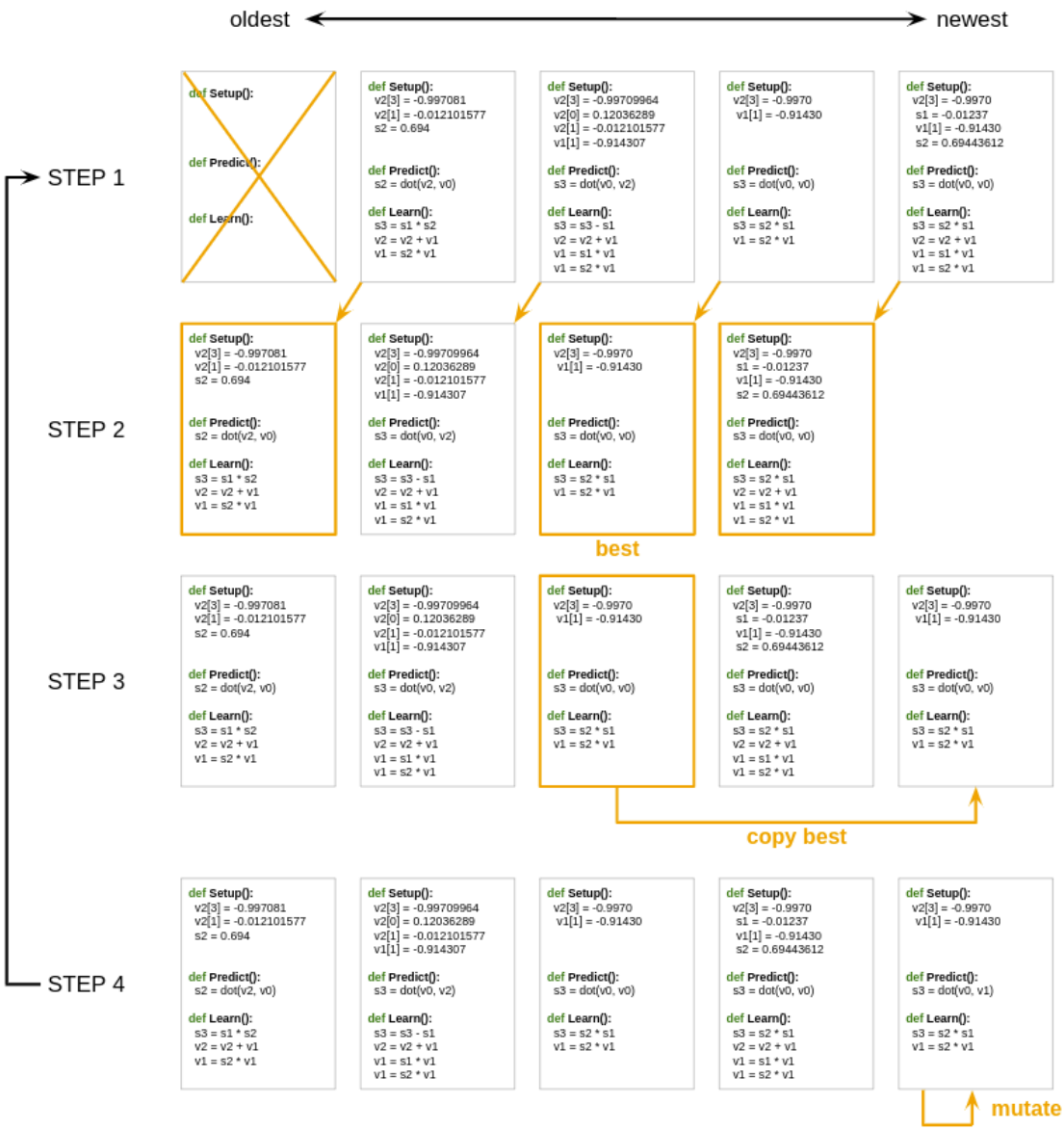

Fig.2.1　RE-AutoML-Zero[11] の世代交代モデル．STEP1 から STEP4 を繰り返すことで最適なアルゴリズムの発見を目指す．STEP1 で最も古い個体を削除した後に, STEP2 でトーナメント選択を行う．その後, STEP3 でトーナメント選択によって選ばれた個体をコピーし, STEP4 で一定確率 p_{mutate} で突然変異を行う．

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．

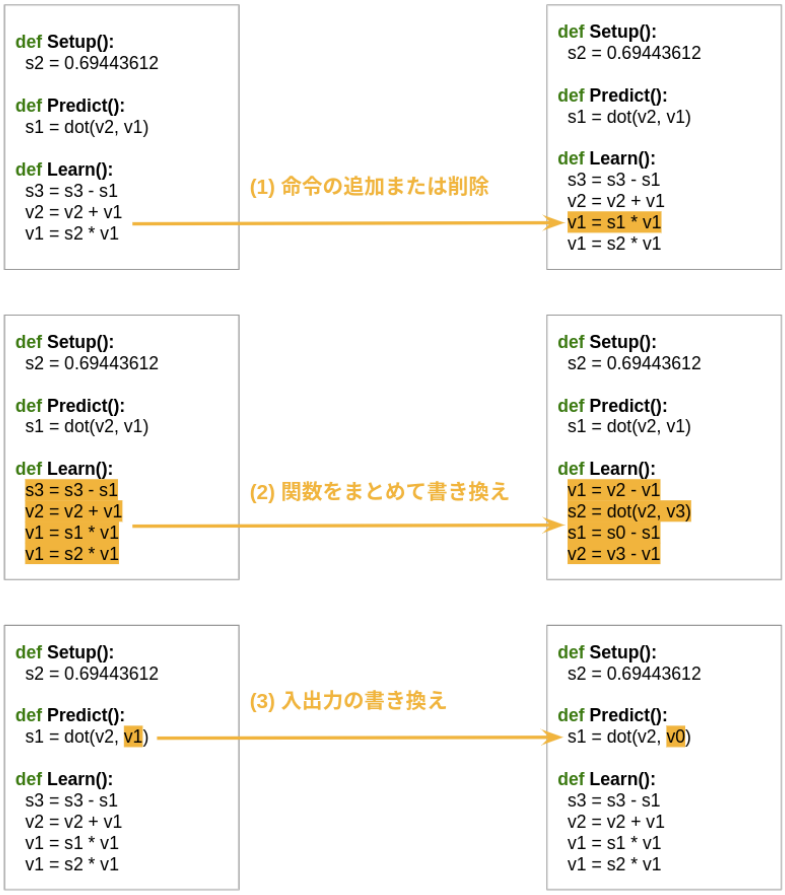

Fig.2.2　突然変異の種類 [11]. (1) アルゴリズムの Setup, Predict, Learn のいずれかをランダムに選択した上で, ランダムに命令を追加または削除する. (2) アルゴリズムの Setup, Predict, Learn のいずれかをすべてランダムな命令列で書き換える. (3) アルゴリズム内のいずれかの命令の入出力をランダムに変更する.

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+VAT

3.1 はじめに

本章では, Esteban らの既存手法 RE-AutoML-Zero に関する問題を指摘した上で, 問題点に対処した手法を提案する. 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, 第 3.4 節では提案手法の有効性の確認するための実験, 第 3.5 節では実験の結果を踏まえた考察を行う. 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, 集団の多様維持に関する問題, 探索空間の冗長性に関する問題, 破壊的な突然変異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.

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.

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が探索対象となる. ここで, 妥当ではないアルゴリズムと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を持っておらず, 探索する必要性がないアルゴリズムを意味する. 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.

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. 実際, 既存手法で線形回帰アルゴリズムを探索する際に生成されるアルゴリズムの 99% 以上が妥当ではないアルゴリズムであることが予備実験により分かっている.

Code. 3.1　妥当な機械学習アルゴリズムの例

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v0)

def Learn():
  s3 = s2 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  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.

```
def Setup():
  s2 = 0.01

def Predict():
  s2 = dot(v1, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  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.

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v3)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  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.

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v3, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  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
```

さらに, 既存手法には変数名と命令の順序による探索空間の冗長性も存在する. Code. 3.9 に挙げた既存手法のアフィン回帰アルゴリズムの表現を例に説明する. Code. 3.9 において, s6 という変数が s8 に変わったり, v6 が v1 に変わっても本質的に同等のアルゴリズムである. また, 20 行目と 21 行目が入れ替わったり, 16 行目と 17 行目が入れ替わっても同等のアルゴリズムである. 既存手法では, これらのアルゴリズムを区別して評価が行われているため, 冗長な探索空間を探索していると考えられる.

Code. 3.9　既存手法のアフィン回帰アルゴリズムの表現

```
def Setup():
  v6[0] = -0.26211548
  v6[1] = 0.6834092
  v6[2] = 0.85052276
  v6[3] = 0.19353867
  s3 = -0.24494825365333672
  s2 = 0.98132917397193
  s7 = 0.3671719126370274

def Predict():
  s6 = dot(v0, v6)
  s1 = s6 + s7

def Learn():
  s4 = s0 - s1
  s6 = s3 * s4
  s5 = s3 * s4
  v3 = s6 * v0
  v6 = v6 + v3
  s7 = s7 + s5
  s3 = s3 * s2
```

3.2.3　破壊的な突然変異に関する問題

既存手法の突然変異には, 程よい局所性を持つ突然変異が存在しないという問題が存在する. 既存手法の突然変異 (1), (2) は, アルゴリズムの構造を大きく変更することはない. しかし, この 2 つの突然変異では大域的な探索を行うことは難しい. 一方で, 突然変異 (3) は, Setup, Predict, Learn のいずれかの関数を全て変更してしまうため, 良質なアルゴリズムの部分構造を淘汰させ, 探索性能の低下を引き起こす可能性がある. したがって, 突然変異 (1), (2) と突然変異 (3) の中間程度の局所性を持つ突然変異が望ましいと考えられる.

3.3　提案手法

本節では, 既存手法である RE-AutoML-Zero の問題点に対処した手法 MGG-AutoML-Zero+VAT を提案する. 探索空間の冗長性の問題に対しては, アルゴリズムの木構造表現 (Valid Algorithm Tree, VAT) と学習パラメータの依存関係を表す有向グラフ (Learning Parameters Network, LPN) によって個体を表現した上で, 妥当ではないアルゴリズムを実行不可能解とすることで対処する. また, 破壊的な突然変異に対しては, 部分木を用いた突然変異手法, 集団の多用維持に対しては, RE の代わりに MGG (Minimal Generation Gap) と呼ばれる世代交代モデルを活用することで対処する.

以下では, 第 3.3.1 節で VAT, 第 3.3.2 節で LPN, 第 3.3.3 節で妥当なアルゴリズムの条件, 第 3.3.4 節で初期個体の生成方法, 第 3.3.4 節で子個体の生成方法, 第 3.3.6 節で MGG による世代交代について述べる.

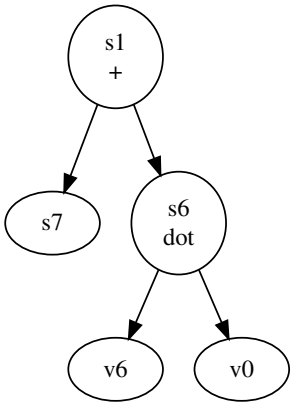

Fig.3.1　Code.3.9 の $s1$ を算出するための木

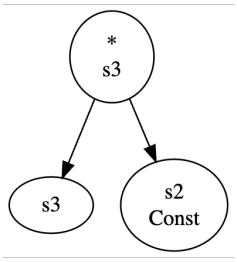

Fig.3.2　Code.3.9 の $s3$ を更新するための木

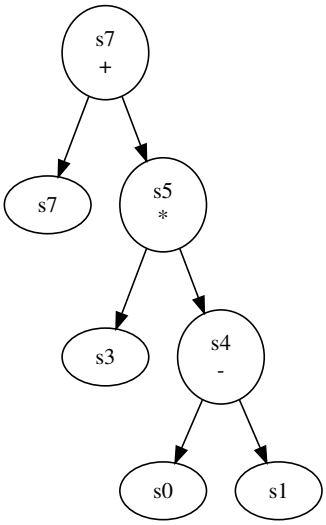

Fig.3.3　Code.3.9 の切片 $s7$ を更新するための木

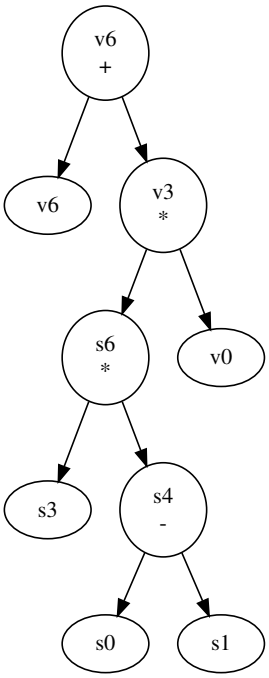

Fig.3.4　Code.3.9 の傾き $v6$ を更新するための木

3.3.1　Valid Algorithm Tree (VAT)

学習率適応を導入したアフィン回帰アルゴリズムと解釈可能な Code.3.9 の VAT は, Fig.3.1-Fig.3.4 に示した複数の木で表される．Fig.3.1 は, 予測ラベル $s1$ を算出するための命令列の木であり Predict 関数に対応する．Fig.3.2 は, 学習率 $s3$ に対応する学習パラメータの更新用の木で, Code.3.9 の Learn 関数内の 18 行目に対応する．Fig.3.3 は, 切片 $s7$ に対応する学習パラメータの更新用の木で, Code.3.9 の Learn 関数内の 12, 13, 13, 14, 17 行目に対応する．Fig.3.4 は, 傾き $v6$ に対応する学習パラメータの更新用の木で, Code.3.9 の Learn 関数内の 12, 13, 15, 16 行目に対応する．

VAT を構成するそれぞれの木では, 葉以外のノードには命令, 葉のノードには, 定数, 学習パラ

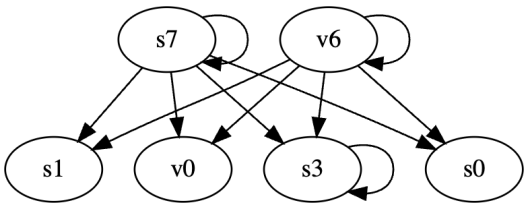

Fig.3.5　Code.3.9 の LPN

メータ, 入力ベクトル $v0$, 正解ラベル $s0$ が割り当て可能である. ただし, $s1$ を算出するための木の葉には, 正解ラベル $s0$ を割り当てることはできない. また, 全てのノードは型（スカラー, ベクトル, 行列）を持ち, 命令が割り当てられているノードには, その命令の入力の型に合う子ノードを割り当てる. 例えば, Fig.3.2 の場合は, ルートノードにはスカラーの乗算命令が割り当てられており, 入力として $s3$ と定数である $s2$, 出力として $s3$ が設定されている. そのため, これらのノードは $s3 = s3 * s2$ という命令と解釈できる.

また, Setup 関数に対応する木は構成しないが, 学習パラメータの初期値や定数値を設定可能にしておくことで代替できることに注意されたい.

3.3.2　Learning Parameters Network (LPN)

Learning Parameters Network (LPN) は, 学習パラメータ, 入力ベクトル $v0$, 予測ラベル $s1$, 正解ラベル $s0$ の間の依存関係を表す有向グラフである. Fig.3.5 に学習率適応を導入したアフィン回帰のアルゴリズムと解釈可能な Code.3.9 の LPN を示す. LPN は依存関係がある変数に対して矢印を繋ぐ. 自己ループの場合は, 1 ステップ前の自分自身に依存していることを意味する. そのため, Fig.3.5 の LPN の場合は, 切片と傾きが入力ベクトル $v0$, 予測ラベル $s1$, 正解ラベル $s0$, 学習率, 1 ステップ前の自分自身に, 学習率は 1 ステップ前の自分自身に依存していることを表している.

3.3.3　妥当なアルゴリズムの条件

提案手法では個体表現方法として VAT と LPN を使用することで, 既存手法で探索対象となっていた妥当ではないアルゴリズムを実行不可能解として扱い, 探索空間から除外する. 具体的には, VAT と LPN に対して以下の条件を満たさないアルゴリズムを実行不可能解とする.

- $v0 \in \mathrm{Leaf}(T_{s1})$
  $s1$ の算出に $v0$ が使われていること.
- $\exists i,\ \mathrm{LP}_i \in \mathrm{Leaf}(T_{s1})$
  $s1$ の算出に直接使われている学習パラメータが存在すること.
- $\forall i,\ \exists j,\ \mathrm{LP}_j \in \mathrm{Leaf}(T_{\mathrm{LP}_i})$
  学習パラメータは 1 ステップ前の学習パラメータを使って更新されること.
- $\forall i,\ \mathrm{LP}_i \in \mathrm{Leaf}(T_{s1})$
  $\Rightarrow \{(\mathrm{LP}_i, v0), (\mathrm{LP}_i, s0), (\mathrm{LP}_i, s1), (\mathrm{LP}_i, \mathrm{LP}_i)\} \subset \mathrm{Path}(G_{\mathrm{LPN}})$
  $s1$ の算出に直接使われる学習パラメータは全て $v0, s0, s1$, 1 ステップ前の自分自身に依存していること.
- $\forall i,\ \exists j,\ \mathrm{LP}_i \notin \mathrm{Leaf}(T_{s1})$
  $\Rightarrow (\mathrm{LP}_j, \mathrm{LP}_i) \in \mathrm{Path}(G_{\mathrm{LPN}}) \wedge \mathrm{LP}_j \in \mathrm{Leaf}(T_{s1})$

すべての学習パラメータが $s1$ の算出に直接的もしくは他の学習パラメータを媒介して間接的に使われていること.

ここで, $\mathrm{Leaf}(T_{s1})$ は $s1$ を算出するための木 T_{s1} の葉ノードの集合, $\mathrm{Path}(G_{\mathrm{LPN}})$ は LPNG_{LPN} 上に存在するパスの始点と終点の順序対全体の集合を表す. また, LP_i は学習パラメータを表す.

また, VAT の導入によって変数名が異なるだけで本質的に同じアルゴリズムの冗長性を排除することができる. 実際, Fig.3.1-3.4 の木では, Code.3.9 との対応関係をわかりやすくするために, 中間ノードにおいても変数名を明示しているが, 実際に動作させるときは変数名を保持する必要がない. 言い換えれば, 変数名が異なるだけで本質的に同じアルゴリズムを区別せずに, 同一個体として扱うことが可能となる.

3.3.4 初期個体の生成方法

提案手法では, VAT, LPN と妥当なアルゴリズムの条件を用いて以下のステップで初期個体の妥当なアルゴリズムを生成する.

1. $v0$ を葉ノードとして持つように, $s1$ を算出するための木 T_{s1} をランダムに生成する.
2. $v0$ と定数ノード以外の T_{s1} の葉ノードを学習パラメータとして取り出す.
3. 取り出した学習パラメータをベースに妥当なアルゴリズムの条件を満たすようにランダムに LPN を構成する.
4. LPN で定義される依存先を葉ノードとして持つように, 各学習パラメータの木をランダムに生成する.

3.3.5 子個体の生成方法

本節では, 既存手法の破壊的な突然変異に対処した新たな突然変異による子個体の生成方法について説明する. 提案手法では, 以下の 4 つの突然変異を導入する.

1. 定数の値をランダムに変更する.
2. 学習パラメータの初期値を変更する.
3. VAT の葉ノードを置き換え可能な別の葉ノードに置き換える.
4. VAT の部分木を変更する.

ここで置き換え可能な別の葉ノードとは, 妥当なアルゴリズムの条件を満たす範囲内で置き換えられる定数, 学習パラメータ, 入力ベクトル $v0$, を意味する. また, 突然変異を行う確率は, 突然変異 (1) が 0.1, 突然変異 (2) が 0.1, 突然変異 (3) が 0.3, 突然変異 (4) が 0.5 である.

以下では, 突然変異 (4) を Fig.3.6 に示した具体例で説明する. 部分木の変更による突然変異は以下の流れで行われる.

1. VAT に含まれる木の中からランダムに 1 つを選択する.
2. 1 で選ばれた木から葉以外のノードを 1 つから 3 つ選択する (Fig.3.6 の青色で囲まれたノード). ただし, 2 つ以上のノードを選択する場合は, 選択されるノードが全て連結している必要がある.
3. 2 で選ばれたノードおよびその子ノード (Fig.3.6 の赤色で囲まれたノード) を合わせた部分木を抽出する.
4. 3 で抽出した部分木を制約条件を満たすように再構成する.

ここで, 4 における制約条件は以下の 2 つである.

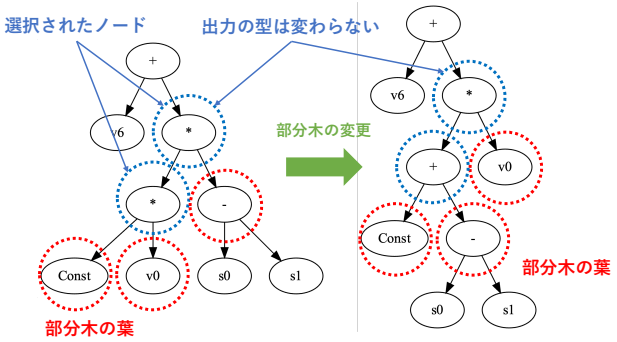

Fig.3.6　VAT の部分木の変更による突然変異

- 部分木のルートノードの出力の型が再構成前と後で変わらないこと
- 再構成前の部分木の葉ノードが, 再構成後の部分木の葉ノードとして存在すること

この制約条件を満たして, 突然変異をさせることで, 抽出された部分木以外の部分を一切変更することなく, 突然変異を行うことが可能になる.

提案手法の突然変異 (4) は, 既存手法の突然変異 (3) よりも局所的な突然変異を実現できると考えられる. 既存手法の突然変異では, 特定の関数に含まれる命令が全て再構成されてしまうことがあった. 一方で, 提案手法では Predict 関数に対応する木を更新しても, 各学習パラメータを更新するための木には影響を与えない. また, 学習パラメータごとに木が独立されて管理されているため, 既存手法のように Learn 関数の一部を変更したら, 複数の学習パラメータに影響を与えてしまうことはない.

さらに, 提案手法の突然変異は既存手法のように妥当なアルゴリズムを破壊することがない. 提案手法の突然変異は, 親が妥当なアルゴリズムであれば子も妥当なアルゴリズムになる. そのため, 既存手法の突然変異のように, 妥当なアルゴリズムの 99.97% 以上が突然変異によって妥当ではないアルゴリズムに変わってしまう問題は発生しない.

以上の理由から, 提案手法では既存手法よりも親個体の良質なアルゴリズムの部分構造を子個体が継承することができ, 探索性能が向上することが期待される.

3.3.6　MGG による集団の多様性維持

提案手法では, 集団の多様性を維持しやすくするため, RE-AutoML-Zero と同様の方法で初期集団を生成した後に, Fig.3.7 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 $N_{\text{children}}/2$ 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.

世代交代モデルを MGG に置き換えることで, 3.2.1 節で述べた 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. MGG では, Fig.3.7 の STEP1 で淘汰される候補となった親個体も, Fig.3.7 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. また, Fig.3.7 の STEP1 で無作為に複製選択するため, トーナメント選択のような強い選択圧がかかりにくいと考えられる. 加えて, 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ため, 淘汰される個体と無

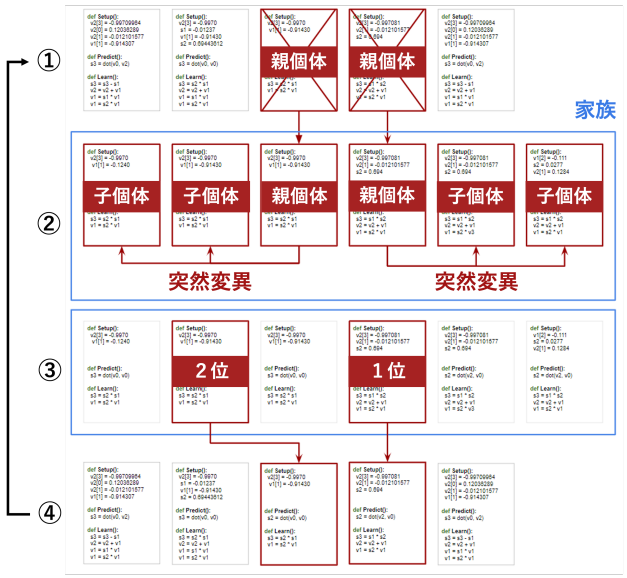

Fig.3.7　MGG による世代交代の流れ

関係な個体が次世代に引き継がれることが少ない.

3.4　実験

3.4.1　目的

本実験の目的は, 線形回帰アルゴリズム, アフィン回帰アルゴリズム, 非線形回帰アルゴリズムの探索問題において, 提案手法の探索性能を既存手法と比較をすることである. 具体的には, 既存手法である Esteban らの手法 RE-AutoML-Zero[11] に比べて, 提案手法の MGG-AutoML-Zero+VAT が, 一定評価回数以内で発見できるアルゴリズムの性能の観点で優れていることを確認する.

3.4.2　ベンチマーク問題

本実験では, 線型回帰アルゴリズムの探索問題, アフィン回帰アルゴリズムの探索問題, 非線形回帰アルゴリズムの探索問題を用いた. それぞれの問題では, 10 件のタスクで構成される探索用のタスク集合 $\mathcal{T}_{\text{search}}$ と 100 件のタスクで構成される評価用のタスク集合 $\mathcal{T}_{\text{eval}}$ を用意する. また, $\mathcal{T}_{\text{search}}$ 内の各タスクの学習用データと検証用データの個数はどちらも 100 に設定した. 一方, $\mathcal{T}_{\text{eval}}$ 内の各タスクの学習用データの個数は 1000, 検証用データの個数は 100 に設定した. 以下では, $\mathcal{T}_{\text{search}}$ と $\mathcal{T}_{\text{eval}}$ に含まれる各タスク $T^{(i)}$ の構成方法を問題設定ごとに説明する.

線型回帰アルゴリズムの探索問題

線型回帰の探索問題では, 各タスク $T^{(i)}$ は線型タスクである. それぞれのタスクの入力ベクトル $\boldsymbol{x}_j^{(i)}$ と正解ラベル $y_j^{(i)}$ は, $y_j^{(i)} = \boldsymbol{w}^{(i)} \cdot \boldsymbol{x}_j^{(i)}$, $\boldsymbol{x}_j^{(i)} \sim \boldsymbol{N}(0,1)$, $\boldsymbol{w}^{(i)} \sim \boldsymbol{N}(0,1)$ を満たす. 問題の次元 $\dim\left(\boldsymbol{x}_j^{(i)}\right)$ は 4 に設定した.

アフィン回帰アルゴリズムの探索問題

アフィン回帰の探索問題では, 各タスク $T^{(i)}$ はアフィン回帰タスクである. それぞれのタスクの入力ベクトル $\boldsymbol{x}_j^{(i)}$ と正解ラベル $y_j^{(i)}$ は, $y_j^{(i)} = \boldsymbol{w}^{(i)} \cdot \boldsymbol{x}_j^{(i)} + u^{(i)}$, $\boldsymbol{x}_j^{(i)} \sim \boldsymbol{N}(0,1)$, $\boldsymbol{w}^{(i)} \sim \boldsymbol{N}(0,1)$, $u^{(i)} \sim N(0,1)$ を満たす. 問題の次元 $\dim\left(\boldsymbol{x}_j^{(i)}\right)$ は 4 に設定した.

非線形アルゴリズムの探索問題

非線形回帰問題では, 各タスク $T^{(i)}$ は中間層が 1 層 4 ノードのランダムなニューラルネットワークで生成した非線形回帰問題である. それぞれのタスクの入力ベクトル $\boldsymbol{x}_j^{(i)}$ と第 k 番目の中間層のノードの出力 $v_{j,k}^{(i)}$ は $v_{j,k}^{(i)} = \mathrm{sigmoid}\left(\boldsymbol{x}_j^{(i)} \cdot \boldsymbol{w}_k^{(i)}\right)$, $\boldsymbol{x}_j^{(i)} \sim \boldsymbol{N}(0,1)$, $\boldsymbol{w}_k^{(i)} \sim \boldsymbol{N}(0,1)$ を満たす. また, 中間層の出力 $\boldsymbol{v}_j^{(i)} = \left(v_{j,1}^{(i)}, v_{j,2}^{(i)}, v_{j,3}^{(i)}, v_{j,4}^{(i)}\right)$ と正解ラベル $y_j^{(i)}$ は, $y_j^{(i)} = \mathrm{sigmoid}\left(\boldsymbol{v}_j^{(i)} \cdot \boldsymbol{w}_{\mathrm{out}}^{(i)}\right)$,$\boldsymbol{w}_{\mathrm{out}}^{(i)} \sim \boldsymbol{N}(0,1)$ を満たす. 問題の次元 $\dim\left(\boldsymbol{x}_j^{(i)}\right)$ は 4 に設定した.

3.4.3 評価基準

本実験の性能評価指標としては, 所定の評価回数に到達するまでに発見されたアルゴリズムの最良評価値を用いる. 所定の評価回数は探索問題ごとに定められており, 線型回帰では 50,000 回, アフィン回帰では 2,000,000 回, 非線形回帰では 50,000,000 回とした. また, 評価値には 1 に示した方法で計算した評価値を用いた. 第 3.4.2 で述べた今回のベンチマーク問題では, それぞれ線型回帰アルゴリズム, アフィン回帰アルゴリズム, 中間層が 1 層で活性化関数が sigmoid 関数のニューラルネットワークが発見できた場合に最適解となり, 評価値が 1.000 となる. 最良評価値の比較は, 乱数を変えて 10 試行分行った時の平均で行った.

3.4.4 実験設定

既存手法では, 集団サイズを 1200, トーナメントサイズを 10, 突然変異確率 p_{mutate} を 0.9 とした. 提案手法では, 集団サイズを 300/1200/3600 (linear/affine/nonlinear), 子個体生成数を 200/800/2400 とした. また, 命令数, 命令セット, 変数の個数の上限はそれぞれの最適なアルゴリズムを表現するのに十分になるよう限定して実験を行った.

3.4.5 実験結果

Table.3.2 に既存手法と提案手法で, 線型回帰アルゴリズム, アフィン回帰アルゴリズム, 非線形回帰アルゴリズムの探索を行った結果を示す. Table.3.2 の実験の結果から, すべての問題設定において既存手法に比べて提案手法が高い探索性能を示していることがわかる. 線型回帰とアフィン回帰においては, 既存手法が全ての試行で最適解を発見できなかったのに対して, 提案手法ではすべての試行で最適解の発見に成功した. また, 非線形回帰に関しても, 7/10 で最適解を発見することに成功しており, 既存手法よりも平均して 0.52 以上も高い評価値のアルゴリズムを発見できた. これは, 提案手法が既存手法よりも, 探索空間の削減, 多様性維持, 突然変異おける良質な部分構造の継承が効果的に行えたことを示唆している.

Table.3.2　既存手法と提案手法で, 線型回帰アルゴリズム, アフィン回帰アルゴリズム, 非線形回帰アルゴリズムの探索を行った結果. 行は試行番号, 列は問題設定と手法を表す. 値は所定の回数に到達するまでに発見された最良アルゴリズムの評価値である. 各試行で評価値が高い方を太字で示している.

No.	線形回帰問題		アフィン回帰問題		非線形回帰問題	
	既存手法	提案手法	既存手法	提案手法	既存手法	提案手法
1	0.689	**1.000**	0.412	**1.000**	0.352	**1.000**
2	0.581	**1.000**	0.321	**1.000**	0.374	**1.000**
3	0.489	**1.000**	0.435	**1.000**	0.562	**0.813**
4	0.713	**1.000**	0.357	**1.000**	0.432	**0.882**
5	0.873	**1.000**	0.394	**1.000**	0.452	**1.000**
6	0.819	**1.000**	0.458	**1.000**	0.324	**1.000**
7	0.671	**1.000**	0.314	**1.000**	0.463	**1.000**
8	0.812	**1.000**	0.342	**1.000**	0.347	**1.000**
9	0.728	**1.000**	0.471	**1.000**	0.491	**0.795**
10	0.563	**1.000**	0.391	**1.000**	0.412	**1.000**
Avg.	0.694	**1.000**	0.390	**1.000**	0.421	**0.949**

3.5　考察

3.5.1　提案手法で発見されたアルゴリズム

この節では, 提案手法によって非線形回帰検索で実際に発見された最適なアルゴリズム（Code.3.10）を紹介し, そのアルゴリズムの解釈を提供します. 可読性を向上させるため, 挙動が変わらない範囲でフォーマットされています.

このアルゴリズムは, 1 つの中間層と ReLU 関数を活性化関数として用いたニューラルネットワークとして解釈することができます. 予測関数では, 中間層の重みが 5 行目で適用され, ReLU 関数が 6 行目で適用され, 出力層の重みが 7 行目で適用されます. 学習関数では, 誤差逆伝播が適用されます. この問題設定において, 中間層の重み $\boldsymbol{w}k$ および出力層の重み $\boldsymbol{w}\mathrm{out}$ の誤差逆伝播による更新は以下の通りです.

$$
\begin{aligned}
\boldsymbol{w}_{\mathrm{out}} &\leftarrow \boldsymbol{w}_{\mathrm{out}} - \eta \frac{\partial E}{\partial \boldsymbol{w}_{\mathrm{out}}} \\
\frac{\partial E}{\partial \boldsymbol{w}_{\mathrm{out}}} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{w}_{\mathrm{out}}} = (\hat{y} - y)\boldsymbol{v}, \\
\boldsymbol{w}_k &\leftarrow \boldsymbol{w}_k - \eta \frac{\partial E}{\partial \boldsymbol{w}_k} \\
\frac{\partial E}{\partial \boldsymbol{w}_k} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v_k} \frac{\partial v_k}{\partial u_k} \frac{\partial u_k}{\partial \boldsymbol{w}_k} \\
&= (\hat{y} - y)\ w_{\mathrm{out},k}\ \mathrm{heaviside}(u_k)\ \boldsymbol{x},
\end{aligned}
$$

Code. 3.10　提案手法の非線形回帰アルゴリズム探索で実際に発見された最適なアルゴリズム

```
def Setup():
  s3 = 0.073 // Initialize learning rate.

def Predict():
  v1 = dot(m0, v0) // Apply middle layer weight to input vector.
  v3 = maximum(v1, v2) // Apply ReLU (v2 is zero vector).
  s1 = dot(v3, v4) // Apply final layer weight.

def Learn():
  s2 = s0 - s1 // Calculate error.
  s4 = s2 * s3 // Apply learning rate.
  v5 = s4 * v3 // Calculate update for final layer weight.
  v6 = s4 * v4 // Multiply by the scalar.
  v7 = heaviside(v1) // Apply heaviside to middle layer outputs.
  v8 = v7 * v6 // Take element-wise product.
  m1 = outer(v6, v0) // Calculate update for middle layer weights by outer
       product.
  v4 = v4 + v5 // Update final layer weight.
  m0 = m0 + m1 // Update middle layer weight.
```

3.5.2　MGG の有効性

MGG の有効性を確認するために, 本研究では既存手法の世代交代モデルを MGG に置き換えただけの手法を考察手法として, 線型回帰アルゴリズムとアフィン回帰アルゴリズムの探索実験を行った. 評価回数の上限を 10,000,000 回にして実験をした結果, 既存手法の最適解の発見率が 5/10 だったことに対し, 考察手法では 9/10 であった. この結果から, MGG を導入することで多様性維持が可能となり, 局所解に陥りにくくなったと考えられる.

3.5.3　非線形回帰の失敗理由

本研究では, 提案手法が非線形回帰の探索問題において, 最適解の発見に失敗している原因を実際に集団内のアルゴリズムを観察することによって調査を行った. 調査によって, 学習パラメータが多い LPN を持つアルゴリズムは探索に時間がかかるため, 先に他の単純な LPN を持つアルゴリズムが局所解に収束することが分かった. その結果, 失敗試行では探索過程で最適解と同じ LPN の構造を持つ個体が十分に探索される前に淘汰されていることがわかった.

第 4 章

MGG-AutoML-Zero+AG

4.1 はじめに

本章では, MGG-AutoML-Zero+VAT の問題点を指摘して, その問題点に対処した手法を提案する. 以下, 第 4.2 節では RE-AutoML-Zero の問題点の詳細, 第 4.3 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 MGG-AutoML-Zero+VAT, 提案手法は MGG-AutoML-Zero+VAG を指す.

この提案手法は, 昨日アルゴリズムを再考している過程で思いついたアイディアなので, 今後の課題の詳細ぐらいの位置付けで見て欲しい.

4.2 MGG-AutoML-Zero+VAT の問題点

既存手法の MGG-AutoML-Zero+VAT には, LPN の複雑性に関する問題と部分命令列の関数化ができない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.

4.2.1 LPN の複雑性に関する問題

既存手法では, LPN を構築しているが, 改めてアルゴリズムを再考したところ LPN を構築しなくても, 本質的な探索空間に違いがないことが判明した. 本節では, その点について詳細に説明する.

既存手法で LPN を構築していた理由は, 他の学習パラメータを介した間接的な学習パラメータの更新を考慮するためである. 直接的な依存関係は, 各学習パラメータを更新するための木の葉に関する条件のみを規定すれば十分である. 一方で間接的な依存関係は, 学習パラメータを更新するための木の葉に他の学習パラメータ LP_i が存在した場合に, LP_i を媒介した依存関係まで調べる必要がある. LPN を構築せずに, その依存関係を動的に判断しながら, 命令列の木を構築するのは困難である. そのため, 既存手法では間接的な依存関係を含めて, 先に妥当なアルゴリズムの条件を満たすように LPN で依存関係を決定していた.

しかし, アルゴリズムを再考した結果, 間接的な依存関係に関する考慮は本質的に不要であることがわかった. 具体的には, 間接的な依存関係を考慮して判定する妥当なアルゴリズムの条件は, 以下のように直接的な依存関係に書き換えても本質的な探索空間を減らす可能性はないと考えられる.

書き換え前 $\forall i,\ \mathrm{LP}_i \in \mathrm{Leaf}(T_{s1}) \quad \Rightarrow \{(\mathrm{LP}_i, v0), (\mathrm{LP}_i, s0), (\mathrm{LP}_i, s1)\} \subset \mathrm{Path}(G_{\mathrm{LPN}})$
$s1$ の算出に直接使われる学習パラメータは全て $v0, s0, s1$, 1 ステップ前の自分自身に依存していること.

書き換え後 $\forall i,\ \mathrm{LP}_i \in \mathrm{Leaf}(T_{s1}) \quad \Rightarrow v0, s0, s1, \mathrm{LP}_i \in \mathrm{Leaf}(T_{\mathrm{LP}_i})$
$s1$ の算出に直接使われる学習パラメータは全て $v0, s0, s1$, 1 ステップ前の自分自身に直接的に依存していること.

この条件の書き換えが適切であるかを判断するには, $s1$ を算出するための木で利用されている学習パラメータに関して, 以下の 3 つのアルゴリズムが探索不要であることを示れば良い.

- 直接的な依存関係に $v0, s0, s1$ の一部と 1 ステップ前の自分自身のみが含まれている場合
- 直接的な依存関係に 1 ステップ前の自分自身が含まれておらず, 他の学習パラメータを介して間接的に自分自身に依存している場合
- 直接的な依存関係に $v0, s0, s1$ のいずれも含まなておらず, 他の学習パラメータを介して間接的に $v0, s0, s1$ に依存している場合

1 つ目は, 学習パラメータを更新する際に入力, 出力, 正解が同時に存在しない状態で更新することになるため, 妥当な更新ができないと考えられる. 2 つ目と 3 つ目は, 学習パラメータの更新処理を遅延させているだけであり, 直接的な依存関係を用いて同等なアルゴリズムを構築可能である. 妥当なアルゴリズムの条件は, あくまでも最低限満たすべき条件であるため, $v0, s0, s1$ および数学ステップ前の自分自身と間接的な依存関係を持つことを否定するものではないことも合わせて注意されたい.

4.2.2 部分命令列の関数化ができない問題

第 3.5 節で述べたように, 既存手法の非線形回帰アルゴリズムの探索では, 複雑な LPN を持つアルゴリズムの学習が遅くなることに起因して失敗することがある. この探索が遅れる原因として, 学習パラメータの更新をするための木を作る過程で, $s1$ を算出過程や学習パラメータの更新の過程で得られる途中の計算結果が利用できないことが考えられる. Code.3.10 で見つかったアルゴリズムを読むと, 6 行目の Predict 内で代入された変数が Learn 関数の入力して使われていたり, Learn 関数内の 11 行目で代入された変数 $s4$ が複数の箇所で利用されている. [*1]既存手法で, このアルゴリズムを発見する際は, それぞれの学習パラメータを更新するための木で, 完全にゼロから命令列を探索する必要があり, 他の部分で使われている値を再利用することができない. これによって, 探索性能が低下していると考えられる.

4.3 提案手法

文章の整理が追いついていないので, 基本的な考え方を書いて説明します. 概要は以下に示す通りです. 詳細なアルゴリズムについては検討段階である.

- LPN の生成プロセスを廃止して, 各学習パラメータを更新するための木に直接的な依存関係のみで生成できるようにする.
- 木構造を廃止して新たな命令列を管理する用のグラフ構造を定義する. その構造では、葉に他の部分で使われている途中結果を割り当てられるようにする.

[*1] Code.3.10 のアルゴリズムは整形しているため, 実際に生成された結果で共通する処理などはまとめていることに注意されたい.

第 5 章

今後の課題

提案手法は, 昨日アルゴリズムを再考していて思いついた段階なので, 仕様を整理して実装していきたい. また, 既にある MGG-AutoML-Zero+VAT でもある Google の AutoML-Zero と比較して, 1000 倍程度の性能になっているため, 評価の並列化を実装して Esteban らの AutoML-Zero の実験を新しい手法で再現していきたい. そして, 最終的には Esteban らの AutoML-Zero の論文でも試していなかった問題設定についても実験していきたい.

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	-	$s_b = \Vert \boldsymbol{v}_a \Vert$
OP35	$v4 = \mathrm{norm}(m7, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\Vert M_a^{(i,\cdot)} \right\Vert$
OP36	$v4 = \mathrm{norm}(m7, \mathrm{axis} = 1)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\Vert M_a^{(\cdot,i)} \right\Vert$
OP37	$m9 = \mathrm{transpose}(m3)$	a/matrix	-	b/matrix	-	$M_b = M_a^T$
OP38	$m1 = \mathrm{abs}(m8)$	a/matrix	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = \left\vert M_a^{(i,j)} \right\vert$
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.