2023年度

卒業論文

論文題目

Weight Agnostic Neural Networks における活性化関数の慎重な選択

研究者

2031133 増田 瑞樹

指導教員

山口 智 教授

2024年12月26日

目 次

1 はじめに **1**

2 ニューラルネットワーク **3**

2.1 ニューラル素子 3

2.2 活性化関数 5

3 遺伝的アルゴリズム **6**

3.1 本論文における基本的動作 6

3.2 ルーレット選択 8

4 Weight Agnostic Neural Networks **9**

4.1 変異 12

5 提案手法 **13**

5.1 活性化関数同士の区間積分差 15

5.2 個体の経験に基づく出力差 17

5.3 用いる活性化関数 20

5.4 距離関数の妥当性 21

6 実験 **25**

6.1 実験環境 25

6.2 実験内容 26

6.3 実験結果 27

謝辞 **28**

1　はじめに

生物学における先天的能力 (Precociality) とは，動物が生まれた瞬間からすでに持っている能力のことである．例えば，トカゲやヘビは生まれながら捕食者から逃れる能力を有している．また，アヒルは孵化後すぐに泳いだり食事をすることができ，七面鳥は一度も見たことがない捕食者を視認するすることができる．これは，動物の脳は高度に構造化された状態で生まれ，その構造はゲノムに記憶されていることを意味する．Zador は生物学的な学習について『動物の行動の多くは生得的なものであり学習によって生じるものではない．動物の脳は AI 研究者が思い描くようななんでも学べる汎用的な学習アルゴリズムを備えた白紙の状態ではない.』と強調している [1].

Weight Agnostic Neural Networks(WANN) は，2019 年に Adam Gaier と David Ha によって発表されたシナプス荷重に依存しないネットワークの探索アルゴリズムである [2]．WANN は，NEAT[3] をベースに作られており，どのようなシナプス荷重においてもタスクを解ける性質をもつネットワーク，つまり構造自体にタスクを解く機能が備わっている．これは遺伝的アルゴリズム [4] を用いた Neuro-Evolution[5] の手法から実現できる．

WANN の個体変異の 1 つに，ノードの活性化関数の変更が行われる．隠れ層からランダムに選択されたノードの活性化関数は現在採用されている以外の活性化関数へランダムに変更される．これは探索後期において，それまでの良かったノードの出力を反転させてしまう懸念がある．

本論文では，活性化関数を変更する際の確率を関数同士の距離関数が小さいほど選ばれやすいようにする手法を提案する．距離関数が小さいことは，関数同士が似ていることを意味し，活性化関数の変更により出力の大きな変更が起こりにくくなる．距離関数には活性化関数同士の出力の差を積分と，実際にその個体が体験した入力ノードから推測できる該当ノードの出力の差の合計を採用する．得られた距離に ε を加え逆数を取った値をルーレット選択することで活性化関数の変更前の出力と変更後の出力を小さくし，それまでの

良い出力を著しく損ねることを緩和する.

実験では **ここから追加**

2　ニューラルネットワーク

ニューラルネットワークとは機械学習のアルゴリズムで，人工的に構築された神経細胞を模したネットワークからなる．入力層，中間層，出力層から構成され，各層のニューロンは重み付きの結合を通じて信号を伝える [6]．データの特徴を抽出し，分類，予測，生成などのタスクに適用され，画像認識，自然言語処理，音声認識など多岐にわたる応用がある．

2.1　ニューラル素子

人間を含む生命の脳を構成する神経細胞はニューロンと呼ばれ，人間の脳には 140 億個のニューロンがあり 1 つのニューロンは平均約 8,000 個のシナプスを持つとされている．ニューラルネットワークのシナプスとノードはニューロンをモデルとし，計算機上でニューロンをシミュレートできるよう設計されている [6].

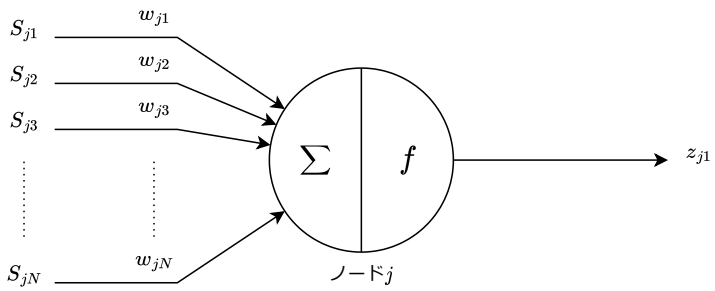

図 1: シナプスとノードの関係

ここでは，ノード j への N 個の入力 $S_{1j}, S_{2j}, S_{3j}, ..., S_{Nj}$ に対して各々の重み $w_{j1}, w_{j2}, w_{j3}, ..., w_{jN}$ となっている．この素子は入力とバイアス b_j を足した値 U_j を活性化関数 f_j の入力とし，活性化関数の出力をノードの出力 z_j とする．

$$U_j = \sum_{i=1}^{N} S_{ji} w_{ji} + b_j \tag{1}$$

$$z_j = f_j(U_j) \tag{2}$$

このノードを組み合わせ，図 2 のような階層型ニューラルネットワークを考える．

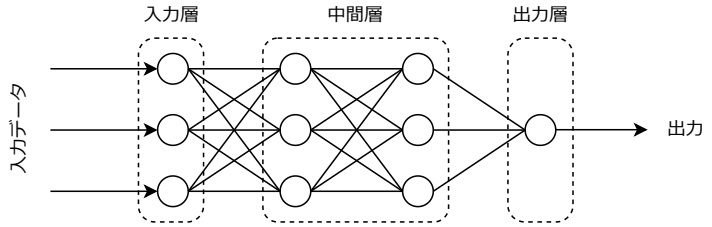

図 2: 階層型ニューラルネットワーク

階層型ニューラルネットワークは，入力層，中間層，出力層からなり，まずタスクを解くための判断材料となる入力データが入力層のノードに入力される．中間層の入力はひとつ前の層の出力からなり，出力層の結果がネットワークの解答になる．

2.2　活性化関数

活性化関数は，与えられた入力をどのように活性化するかの動作を変更するものである．線形分離によって解くことができない問題に対しては，この活性化は非線形変換である必要がある [7]．具体的には実数空間 V, W において V から W への写像 f が以下の性質を満たさない変換である必要がある．ただし $x, y \in V, c \in \mathbb{R}$ とする．

1. 加法性: $f(x+y) = f(x) + f(y)$
2. 斉一次性: $f(cx) = cf(x)$

複数のノードの出力の合計をあるひとつノードとみなすことは線形変換に過ぎず，線形変換の繰り返しはその結果もまた線形性が保たれ，活性化関数を持たないネットワークは隠れ層を持つ場合と持たない場合で表現できる幅は変わらない．複数のニューロンからの入力の合計を活性化関数を通すことはネットワークがより多くの情報を表現できることを意味する．代表的な活性化関数 f は次のようなものがある．

1. ReLU 関数

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{3}$$

2. tanh 関数

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

今回の実験では，これら 2 種類を含む 10 種類の活性化関数を用いる．

3　遺伝的アルゴリズム

遺伝的アルゴリズム (GA) とは，適用範囲の非常に広い，生物の進化を模倣した学習的アルゴリズムである [4]．すなわち何万年もかかる進化の過程を生物の特徴が進化してきたような遺伝的法則を工学的な手法にモデル化し，また参考にしてタスクを解くものである．自然界における生物の進化過程においては，ある世代を形成している個体の集合，すなわち個体群の中で環境に適している個体が高い確率で生き残り，生き残った個体は交叉や突然変異によって次の世代の個体群が形成される．

3.1　本論文における基本的動作

本論文において，新しく個体を形成する際に交叉は使用しない．以降本論文で用いる遺伝的アルゴリズム変則 GA と呼ぶ．図は変則 GA の動作を流れを表している．

1. 初期化

   ランダムな性質を持つ個体を N 個生成し，初期世代の個体群を設定する．

2. 評価

   各個体についてタスクを解かせ，その進捗により適応度を測定する．

3. 終了判定

   終了条件を満たしていれば，その時に得られる最適個体を問題の準最適解として出力する．

4. 選択

   各個体に対応する適応度により並べ替え，適応度の低い個体は淘汰され，適応度の高い個体は増殖する．

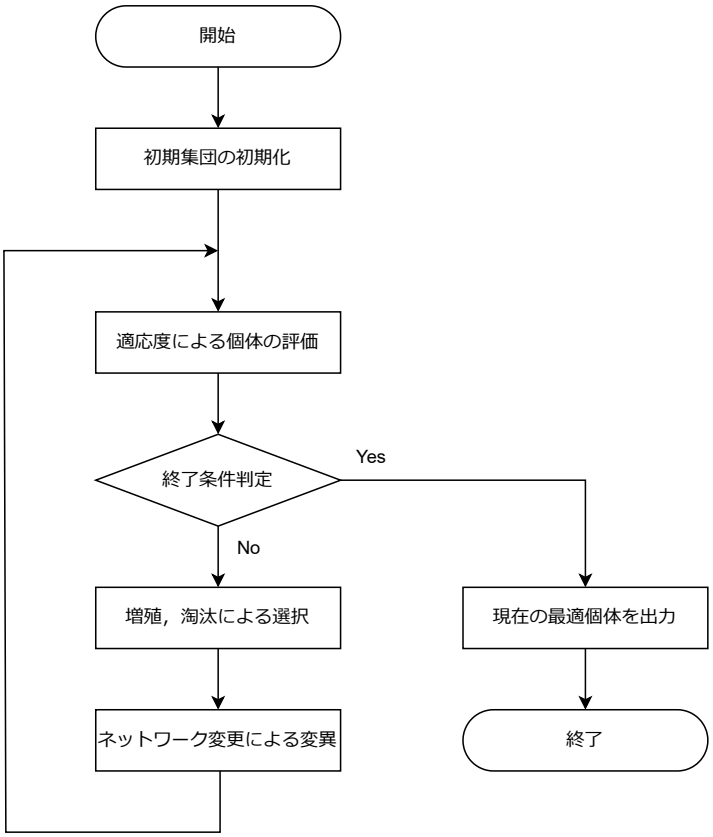

図 3: 変則 GA の概要

5. 変異

設定された突然変異の設定により変異を行い，新しい個体群を生成する．変異を行った後の個体群は次世代の個体群として再度評価される．

終了判定には最良個体の適応度や個体群の平均の適応度を参照する場合が多いが，本論文ではあらかじめ設定した世代数を超えたときにのみ終了判定が真になり，プログラムは終了する．また，変異をする過程でエリート保存選択を適用する．これは非常に優れた個体は変異をする前の状態のほうが変異をした後の状態よりも優れている見込みが大きいことから，個体に対して変異を行わず，全く同じ状態の個体を次世代に残す手法である．エ

リート保存選択は，むやみに変異をして優良個体の遺伝子を破壊しないことにつながる．

3.2　ルーレット選択

ルーレット選択は，個体群の中の各個体の適応度とその総計を求めて，適応度の総計に対する各個体の割合を選択確率として個体を選択するという基本的な考えに基づいている [4]．すなわち，ルーレット選択では，個体 s_i の適応度 $f(s_i)$ と個体群の総計を求め，個体 s_i が選択される確率を

$$P_i = \frac{f(s_i)}{\sum_{j=1}^{N} f(s_j)} \tag{5}$$

として個体の確率的に選択する．このようなルーレット選択のアルゴリズムは，次のように要約される．

1. 世代 t の個体群 $X(t)$ の中の N 個の個体の適応度 f_i とその総計 $f_{sum} = \sum_{i=1}^{N} f_i$ を求める．
2. 区間 [0, 1] の乱数 rand を発生させ， $s = \mathrm{rand} \times f_{\mathrm{sum}}$ とする．
3. $\sum_{i=1}^{k} f_i \geq s$ となるような最小の k を求めて， k 番目の個体を世代 $t+1$ に生き残る個体の候補とする．
4. 候補となる個体数が N になるまで 2, 3 を繰り返す．

本論文ではルーレット選択を個体の変異先を選択する際に用いる．同じ構造を持つネットワークの，ひとつのノードの活性化関数のみを変更し，この出力が小さいほど f_i が大きいことになる．

4 Weight Agnostic Neural Networks

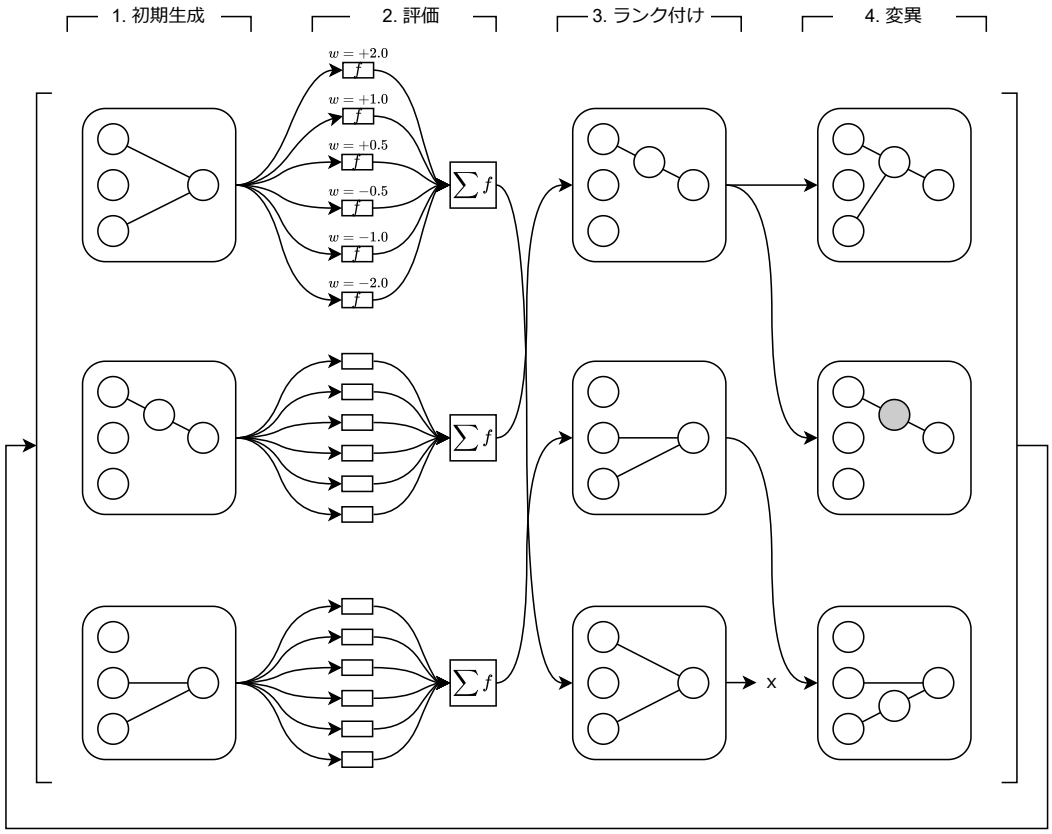

図 4: Weight Agnostic Neural Networks (WANNs) の概略図

Weight Agnostic Neural Networks(WANNs) は，2019 年に Adam Gaier と David Ha によって発表されたニューラルネットワークの構造探索アルゴリズムである [2]．WANN は，NEAT[3] をベースに作られており，これは遺伝的アルゴリズム [4] を用いた Neuro-Evolution[5] の手法から実現できる．多くのニューラルネットワークはシナプス荷重を更新することで，ネットワークの精度を向上させているが，WANNs によって生成された個体は，ネットワーク構造自体がタスクを解く性質を持っており，どんなシナプス荷重においてもそれなりの精度でタスクを解くことができる．これは生物の先天的能力と似た性質であると言える [1].

また，WANNs ネットワークはそれぞれのノードが別々の活性化関数を持っている．WANN の概略図を図 3 に示す．WANN における探索は以下の動作によって実現する．

1. 初期生成個体は最初，最も単純な構造を持つネットワークからなる．中間層は存在せず，入力層のうちの一部からシナプスが出力層に接続されている．

2. 評価各個体に対して，ネットワーク全体の共有重みを使用してタスクを実行する．本論文では，$-2.0, -1.0, -0.5, +0.5, +1.0, +2.0$ の 6 つの共有重みを使用する．また，タスクを解く際の初期状態により優れた結果を残せずに誤った評価をしないよう，それぞれの共有重みで 4 回のタスクを実行する．最終的に 6 つの共有重みと 4 回の試行から得られた 24 個の評価値を合計する．

3. ランク付け個体をランク付けする．上位の優れた個体は変更のないまま次世代へ保存され，下位の劣った個体は淘汰される．このランク付けはネットワークの評価値とネットワークの複雑さから算出される．

4. 変異個体をランクのトーナメント選択 [4] によって選択し，変異を起こした次世代の個体として保存する．

   保存された個体は再度評価，ランク付け，変異を行い，任意の世代数まで繰り返される．

ニューラルネットワークには，その用途を達成するために適したネットワーク構造がある．画像認識には畳み込みニューラルネットワーク (CNN)，自然言語処理には再起型ニューラルネットワーク (RNN) などを用いることで全結合型ニューラルネットワークよりも少ない計算量でよりよい精度を実現することができる [6]．自然界においても海には魚類，陸上には哺乳類などある環境下に適した身体的特徴や先天的能力をさまざまな生物が獲得している [1]．ラットは生まれてから経験したことを，少なくとも小脳皮質のシナプスを追加す

ることによってよりよいフィードバックができるように学習する [10]．WANNs はあるタスクにおいてのみ優れた精度を持つニューラルネットワークを探索するアルゴリズムで，今までのネットワーク構造ありきのシナプス重みを重視する手法とは違い，どんな重みであっても動作するネットワーク構造に着目した手法であり，生物の基本的な性質を踏襲したものになっている．結果として，十分に世代を重ねたときの最良個体の出力は，0 付近を除いたすべての共有重みを用いた際に安定している．

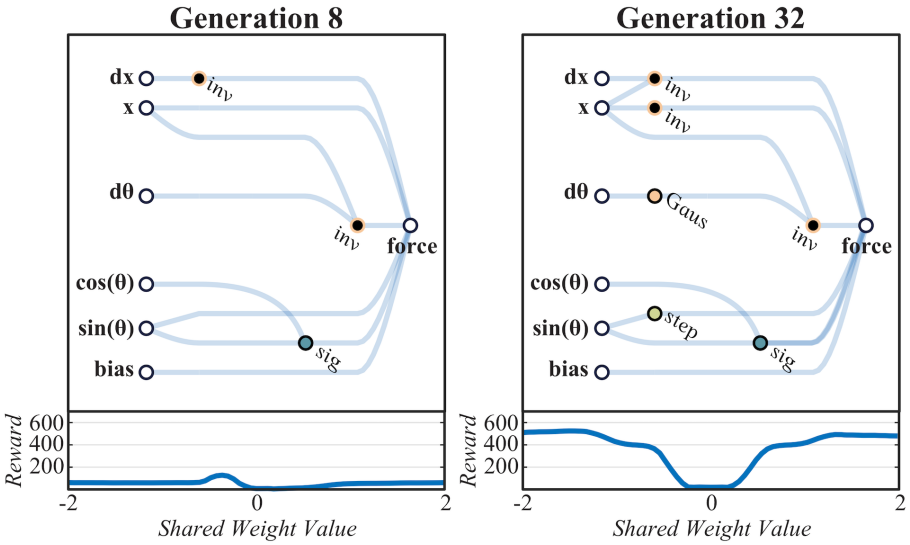

図 5: WANNs のネットワーク構造と共有重み

4.1 変異

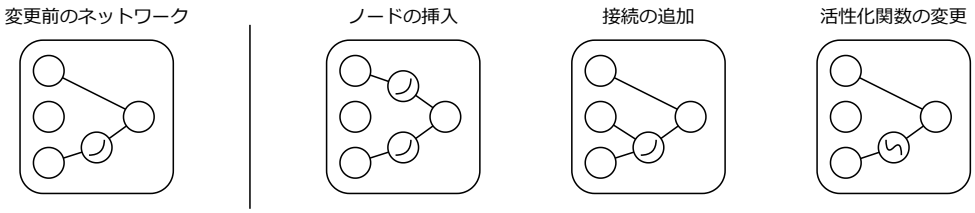

図 6: WANNs のネットワーク変異

個体の変異にはノードの挿入，接続の追加，活性化関数の変更のいずれかを行い，各操作の具体的な手順は以下のようになる．

1. ノードの挿入ネットワークの持つひとつの接続をランダムに選択し，接続の途中にひとつのノードを追加する．この時の活性化関数はランダムに選択される．
2. 接続の追加接続を持たないふたつのノードを選択し，それらをつなぐ接続を追加する．
3. 活性化関数の変更ネットワークの持つ中間層のノードを選択し，そのノードの活性化関数を変更する．活性化関数は現在選択されている関数以外の関数がランダムに選択される．このときの活性化関数は linear, step, sin, cosine, Gaussian, tanh, sigmoid, inverse, absolute value, ReLU を利用する．

5　提案手法

既存手法の問題点として，ランダムな活性化関数の変更が挙げられる．探索後期において良いノードの出力の個体が生まれたとしても，例えば良いノードの活性化関数を linear 関数 $f(x) = x$ から inverse 関数 $f(x) = -x$ に変更されてしまうと，ノードの出力は反転されてしまう．ネットワークの一部の構造化された部分の出力の大小は，同符号であれば出力層に大きな変化をもたらさないが，出力が 0 より大きいか小さいかは，ネットワークの出力は大きく変わることが知られている [2].

提案手法ではこれの問題を緩和するために，活性化関数の慎重な選択について言及する．2 つの活性化関数の距離を計算し，距離が小さければ小さいほど変更先の活性化関数として選択されやすくなる．

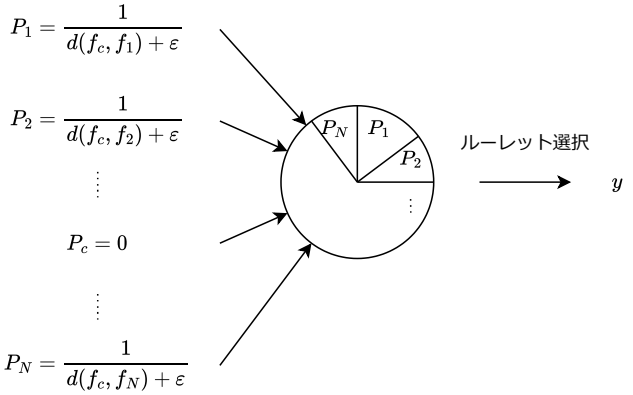

図 7: 提案手法の概略図

$$P_i = \begin{cases} \dfrac{1}{d(f_c, f_i) + \varepsilon_n} & (i \neq c) \\ 0 \qquad (i = c) \end{cases} \tag{6}$$

$$\varepsilon_n = s * \varepsilon_{n-1} \tag{7}$$

表 1: 変数の説明

変数	意味
P_i	c から i へ活性化関数 ID が変更される見込み
$d(f_c, f_i)$	活性化関数が c と i の距離
f_i	ID が i の活性化関数
i	活性化関数 ID
c	現在の活性化関数 ID

まず現在の活性化関数 f_c と，すべての活性化関数との距離を計算する．この時の距離が小さいことは両者の関数が似ていることを意味し，距離の短い関数への変更は既存手法の問題点である出力の反転を防ぐ．距離が小さければ小さいほど選択されやすくするため，距離と ε の和の逆数をルーレット選択の材料とする． ε が大きくなると，ルーレット選択によって選ばれる確率はランダムに近くなる． ε は探索初期においては，良いノードの出力を持つ個体が少ないことから，良い出力の個体を反転させてしまうデメリットより，悪い出力の個体を反転させるメリットの方が大きいと考え，探索初期の ε は大きく，探索後期の ε は小さく設定する．グラフは代表的な活性化関数 4 種，活性化関数同士の区間積分差を用いた活性化関数の変更として選択される確率を表した例である．

ここに図

距離関数 d については，区間積分差と，個体の経験に基づく出力差を採用する．

5.1　活性化関数同士の区間積分差

式 (7) は関数 f_a と f_b の範囲内の出力の差を意味している．

$$d(f_a, f_b) = \int_{-r}^{r} (f_a(x) - f_b(x))^2 dx \tag{8}$$

表 2: 式 (8) の変数の説明

変数	意味
$d(f_a, f_b)$	活性化関数が a と b の距離
f_i	ID が i の活性化関数
r	関数の考慮範囲

実際にネットワークにタスクを解かせる際，ノードへの入力は 0 付近である場合が多い [9]．

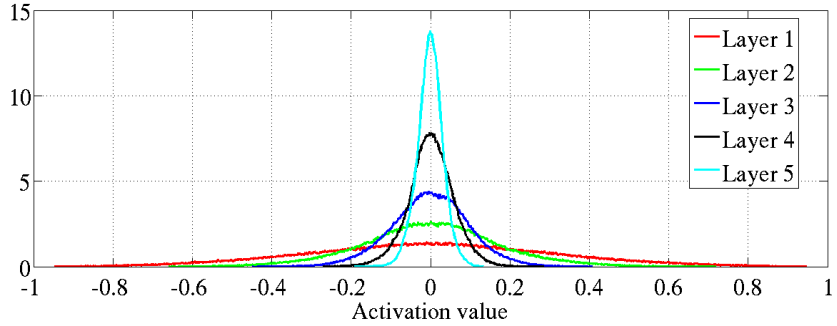

図 8: WANNs のネットワーク変異

このことから考慮する区間は $-r$ から $+r$ までの関数の区間積分によって距離を定義する．区間内の入力に対して出力の差が小さいことは，活性化関数を変更してもネットワークに大きな動作の変更をもたらさないことを意味し，局所的な解を優先的に探索する．番

号が 1 から N と振られている活性化関数 f_1 から f_N ，現在の活性化関数が f_c， f_n と f_c の距離を d_n としたときの具体的なプログラムの実装は以下のようになる．

ソースコード 1: 区間積分差のプログラム

```
for n (1 to N)
    sum = 0
    for x (-r to +r)
        sum += (activate(x, n) - activate(x, c)) ^2
    d[n] = sum
```

¡¡¡¡¡¡¡ HEAD ネットワークで使用する活性化関数にはそれぞれ ID が 1-based（1 からナンバリングする方式）により割り振られており，1 から N までのそれぞれの活性化関数 ID に対してその関数と変更前の関数との差の 2 乗を変数 sum に加算していく．変数 x をとても小さい数ずつ増やしていき， x 範囲内の合計が最終的に sum に格納され，変更前の活性化関数の出力と活性化関数 ID が n の関数の出力の差の 2 乗を d_n に代入する．このように求めた d_n を式 (6) に代入しルーレット選択により選ばれる見込み P_n を得る．また， x を活性化関数の入力とした出力を求める関数 activate は後述する．======= ネットワークで使用する活性化関数にはそれぞれ ID が 1-based（1 からナンバリングする方式）により割り振られており，1 から N までのそれぞれの活性化関数 ID に対してその関数と変更前の関数との差の 2 乗を変数 sum に加算していく．変数 x をとても小さい数ずつ増やしていき，x 範囲内の合計が最終的に sum に格納され，変更前の活性化関数の出力と活性化関数 ID が n の関数の出力の差の 2 乗を d[n] に代入する．このように求めた d_n を式 (6) に代入しルーレット選択により選ばれる見込み P_n を得る．また， x を活性化関数の入力とし， n を活性化関数 ID としたときの出力を求める関数 activate は後述する．¿¿¿¿¿¿¿ d06c96a18ece69aa8a8cf4307e15c26d55166a2f

5.2　個体の経験に基づく出力差

式 (7) は関数 f_a と f_b の個体が経験したノードの入力に対する差を意味している．

$$d(f_a, f_b) = \sum_{m=1}^{M} (f_a(\mathrm{in}_m) - f_b(\mathrm{in}_m))^2 \tag{9}$$

表 3: 変数の説明

変数	意味
$d(f_a, f_b)$	活性化関数が a と b の距離
f_i	ID が i の活性化関数
M	ミニバッチサイズと共有重みの積
in_m	ミニバッチ m 回目に経験したノードに入力される値

区間積分差は範囲内のすべての領域を考慮し差を求めたのに対し，個体の経験に基づく出力差では，実際に個体がタスクを実行したときに経験した入力ノードの値を利用する．ネットワークの構造に変化はないので，入力ノードの値が分かれば，ネットワーク内の全てのノードの出力を特定することができ，ネットワークの変更にあたり指定した隠れ層のノードの出力を入力ノードから計算する．このとき指定したノードの活性化関数のみ変更し，この差を 2 乗を加算することで，個体が経験した入力に対する出力の差の小さい活性化関数を求めることができる．領域内を満遍なく考慮する手法よりも実際の入力ノードを利用することで出現確率の高いノード入力を大きく考慮することを期待する．ミニバッチには，同じ個体であれば異なる共有重みを使用していても利用する．

ソースコード 2: 経験入力に基づく出力差のプログラム

```
1 out(nodeID, actID, state, weight)
2     preout = 0
3     connect[] = synapses where the destination is nodeID
```

```
4     node = Information of node where the ID is nodeID
5
6     if(node[Type] == input)
7         preout = state[nodeID]
8
9     else
10        for i (0 to connect)
11            newnodeID = connect[source]
12            newactID = activationID[connect[source]]
13            val = out(newnodeID, newactID, state[x], weight)
14            preout += val
15
16    prein = preout * weight[m]
17    output = activate(prein, actID)
18    return output
19
20 weight = [-2.0, -1.0, -0.5, +0.5, +1.0, +2.0]
21 for n (1 to N)
22    sum = 0
23    for m (0 to 5)
24        for x (0 to miniBatchSize)
25            sum += (out(ID, n, m, state[x]) - out(ID, c, m, state[x])) ^2
26    d[n] = sum
```

¡¡¡¡¡¡¡ HEAD 1から N までの各活性化関数に対してノードの出力の差を求める．各共有重みに対してミニバッチのうちのひとつの入力ノード情報を利用する．state_x には個体が経験した入力ノード情報が記憶されており，今回使用するタスク BipedalWalker-v2 では，入力層のノードは 24 個なので要素数 24 の配列となる [11]．ノードの出力を求める関数 out には，出力を求めたいノード ID，活性化関数 ID，入力ノード情報，共有重みの値を入力としている．関数 out に入力された nodeID が指すノードが入力層であれば入力ノードデータを返し，隠れ層であれば目的地が nodeID であるすべてのシナプスの 出発地になっているノードに対して出発地のノード ID を newnodeID ，出発地のノードの活性化関数 ID を newactID として out を再帰的に実行する．シナプスの出発地を辿り続けるといつか必ず入力層となるため， val を必ず得ることができる．最初に呼び出した out に入力した nodeID

に入力される値を今までの合計と共有重みの積から計算し，関数 activate に通すことで活性化関数の出力に対応する．最終的に関数 out は nodeID の出力を返し，異なる活性化関数 n と c の差の 2 乗を sum に加算していく．区間積分の手法と同じく n と c の差は d_n に代入される．ソースコード 2 の out(ID, n, m, state[x]) は式 (9) の $f_a(\mathrm{in}_m)$ と同等の意味を持つ． x を活性化関数の入力とした出力を求める関数 activate は後述する．======= 1 から N までの各活性化関数に対してノードの出力の差を求める．各共有重みに対してミニバッチのうちのひとつの入力ノード情報を利用する．state[x] には個体が経験した入力ノード情報が記憶されており，今回使用するタスク BipedalWalker-v2[11] では，入力層のノードは 24 個なので要素数 24 の配列となる．ノードの出力を求める関数 out には，出力を求めたいノード ID，活性化関数 ID，入力ノード情報，共有重みの値を入力としている．関数 out に入力された nodeID が指すノードが入力層であれば入力ノードデータを返し，隠れ層であれば目的地が nodeID であるすべてのシナプスの 出発地になっているノードに対して出発地のノード ID を newnodeID ，出発地のノードの活性化関数 ID を newactID として out を再帰的に実行する．シナプスの出発地を辿り続けるといつか必ず入力層となるため， val を必ず得ることができる．最初に呼び出した out に入力した nodeID に入力される値を今までの合計と共有重みの積から計算し，関数 activate に通すことで活性化関数の出力に対応する．最終的に関数 out は nodeID の出力を返し，異なる活性化関数 n と c の差の 2 乗を sum に加算していく．区間積分の手法と同じく n と c の差は d[n] に代入される．ソースコード 2 の out(ID, n, m, state[x]) は式 (9) の $f_a(in_m)$ と同等の意味を持つ． x を活性化関数の入力とし， n を活性化関数 ID としたときの出力を求める関数 activate は後述する．¿¿¿¿¿¿¿ d06c96a18ece69aa8a8cf4307e15c26d55166a2f

5.3　用いる活性化関数

今回用いる活性化関数には，1-based にナンバリングされ，以下の 10 種類を用いる．また関数への入力 x に対応する出力 y を示した数学的表現，python で実装した入力 x に対応する value を示す.

表 4: 用いる活性化関数とその式

ID	関数名	数学的表現	プログラム上での表現
1	Linear	$y = x$	`value = x`
2	Step	$y = \begin{cases} 1.0 & (x > 0.0) \\ 0.0 & (x \leq 0) \end{cases}$	`value = 1.0*(x>0.0)`
3	Sine	$y = \sin(\pi x)$	`value = np.sin(np.pi*x)`
4	Gaussian	$y = e^{-\frac{x^2}{2}}$	`value = np.exp(-np.multiply(x, x) / 2.0)`
5	Hyperbolic Tangent	$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	`value = np.tanh(x)`
6	Sigmoid	$y = \frac{\tanh\left(\frac{x}{2}\right)+1}{2}$	`value = (np.tanh(x/2.0) + 1.0)/2.0`
7	Inverse	$y = -x$	`value = -x`
8	Absolute	$y = \lvert x \rvert$	`value = abs(x)`
9	Relu	$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$	value = np.maximum(0, x)
10	Cosine	$y = \cos(\pi x)$	`value = np.cos(np.pi*x)`

上で使用した activate 関数は，入力値と活性化関数 ID とし，表に従い value を出力する.

5.4　距離関数の妥当性

提案手法では，活性化数として表現している．

実数の組から実数への写像を実現する関数 $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ が距離関数であるとは， x, y, z を $\mathbb{R}$ の任意の元として，以下の条件が満たされていることを言う [8].

表 5: 距離関数の性質

性質	定義
非負性	$d(x, y) \geq 0$
同一律	$d(x, y) = 0 \Leftrightarrow x = y$
対称律	$d(x, y) = d(y, x)$
三角不等式	$d(x, y) \leq d(x, z) + d(z, y)$

同一律については，$d(x, x) = 0$ とは違う意味を示していることに注意する． x と x の距離は当然 0 であるが， x と y が同一でない限り距離が 0 にはならないことを示している．つまり x と y が別のものを指しているのに関わらず距離が 0 になった場合，その関数 d は距離関数としての性質を満たしていないことになる．これらの証明をするにあたって，2 つの提案手法について確認する．式 (8) と式 (9) はいずれも一定の数の要素を持つ配列の，各要素同士の差を 2 乗したものの総和ととれる．各要素の値はどのようなものであっても距離関数を満たすため，ここでは以下のようにし証明を簡略化する．

$$d(a, b) = \sum_{i=o}^{N-1} (a[i] - b[i])^2 \tag{10}$$

ここで $f_a(x)$ や $f_a(in_m)$ を $a[i]$ とすることは，どのような実数に対しても関数が距離関数の性質を持つことから，そのうちの一例である $f_a(x)$ や $f_a(in_m)$ も例外でなく距離関数の性質を持つと導くことができる．以下は同じ要素数の配列に任意の実数を代入したときに式 (10) が距離関数として成立しているかの証明になる．

1. 非負性

   任意の実数 $a[i]$ と $b[i]$ の差は必ず実数となり，実数の 2 乗は必ず 0 以上の実数になる．有限の個数の 0 以上の実数の和もまた必ず 0 以上の実数となることから，式 (10) は非負正を満たす.

2. 同一律

   $(a[i]-b[i])^2$ が必ず 0 以上の実数となり，0 になるのは $a[i]$ と $b[i]$ の値が等しい場合のみであることから，関数 d が 0 になるのは，すべての要素に対して $a[i]$ と $b[i]$ が等しい必要がある．これは全ての要素について $a[i]=b[i]$ であることを意味し，同一率を満たす.

3. 対称律

   $x^2=(-x)^2$ であることから，$(a[i]-b[i])^2=(b[i]-a[i])^2$ と言える．その総和もまた等しいため，関数 d は対称律を満たす.

4. 三角不等式

   三角不等式の証明にはシュワルツの公式

$$\sum_{k=1}^{n} p_i q_i \leq \sqrt{\sum_{k=1}^{n} p_i^2 + \sum_{k=1}^{n} q_i^2} \tag{11}$$

   を用いる．まず a の各要素を $a_0, a_1, a_2, ..., a_m$ とし，これと同じく b, c についても表す．次に

$$p_i =, q_i = b_i - c_i \tag{12}$$

   とし，これより

$$p_i + q_i = b_i - a_i \tag{13}$$

を得る．対称律，式 (11)，式 (12)，式 (13) を用いると

$$\begin{aligned}
\{d(a,b)\}^2 &= \{d(b,a)\}^2 \\
&= \{\sum_{i=0}^{N-1}(b_i - a_i)^2\}^2 \\
&= \{\sum_{i=0}^{N-1}(p_i + q_i)^2\}^2 \\
&= \{\sum_{i=0}^{N-1}(p_i^2 + 2p_iq_i + q_i^2)\}^2 \\
&= \{\sum_{i=0}^{N-1}p_i^2 + \sum_{i=0}^{N-1}2p_iq_i + \sum_{i=0}^{N-1}q_i^2\}^2 \\
&\leq \{\sum_{i=0}^{N-1}p_i^2 + 2\sqrt{\sum_{i=0}^{N-1}p_i^2\sum_{i=0}^{N-1}q_i^2} + \sum_{i=0}^{N-1}q_i^2\}^2 \\
&= \{(\sqrt{\sum_{i=0}^{N-1}p_i^2} + \sqrt{\sum_{i=0}^{N-1}q_i^2})^2\}^2 \\
&= \{\sum_{i=0}^{N-1}p_i^2 + \sum_{i=0}^{N-1}q_i^2\}^2 \\
&= \{\sum_{i=0}^{N-1}(c_i - a_i)^2 + \sum_{i=0}^{N-1}(b_i - c_i)^2\}^2 \\
&= \{d(a,c) + d(c,b)\}^2 \\
d(a,b) &\leq d(a,c) + d(c,b)
\end{aligned} \tag{14}$$

より三角不等式を満たす.

これらの証明は，いずれも a[i] の要素の集合が実数集合のうちの一部であることに注意する．例えば式 (8) では $-r$ から r までの限られた範囲に過ぎない．式 (9) では経験した入力ノードの組み合わせ，さらにそのミニバッチサイズ分のデータに対しての距離を計算しているに過ぎない．Linear 関数と Relu 関数は x が 0 以上のみの入力を考慮した場合距離が 0 になってしまうが，実際これらの活性化関数は負の領域では全く異なる性質を持つ．しかしこの式は，ネットワークが多くの場面で経験する値を想定して入力として扱っている．0 より大きく離れた値はノードの入力にはめったに使われなかったり，経験した入力ノードの値から大きく外れたものもあまり考慮する必要がないという考え，今回の提案手法を採用した．

6　実験

¡¡¡¡¡¡¡ HEAD 本実験の目標は，変異の際の活性化関数の慎重な選択をすることによってよりよい精度を実現することである．つまり探索後期において，ネットワークの出力が大きく異なる変異を避けることで，局所的な解を重点的に探索できるようにする．また，ε の値により探索初期の収束速度の向上も期待する．======= 本実験の目標は，変異の際の活性化関数の慎重な選択をすることによってより高い精度を実現することである．すなわち，探索後期においてネットワークの出力が大きく異なる変異を回避することで，局所的な解を優先的に探索できるようにする．また，ϵ の値により探索初期の収束速度の向上も期待する．¿¿¿¿¿¿¿ d06c96a18ece69aa8a8cf4307e15c26d55166a2f

6.1　実験環境

以下は本実験を行うにあたり使用した環境である．

表 6: 実験環境

OS	Ubuntu 22.04.3 LTS(64bit)
CPU	Core(TM) i7-12700
GPU	GeForce RTX 3080
RAM	32.0GiB
使用言語	Python 3.8.12

実験を行うにあたって，CPU の性能は個体の変異や選択に要する時間に関係する．今回 12 コアのうち，マスターコアに 1 コア，スレーブコアに 11 コアを割り当てている．GPU の性能は個体がタスクを得く際の適応度の測定時間に関係する．浮動小数点演算には 64bit を用いることが推奨されているが，今回のタスクにおいて大きな誤差が確認できないため，速度の早い 32bit を使用する．

6.2　実験内容

本実験では，既存手法，区間積分差による提案手法，個体が経験した入力に対する出力差による提案手法の 3 つを比較する．また，提案手法については ε の有無についても比較する．以下は上記の 5 つの異なるパラメータを表した表である．

表 7: 比較する実験

実験番号	活性化関数の変更アルゴリズム	ε の初期値	ε の減衰率
1	既存手法	-	-
2	区間積分差を用いた慎重な選択	0.1	1.0
3	区間積分差を用いた慎重な選択	1.0	0.99
4	個体の経験に基づく出力差を用いた慎重な選択	0.1	1.0
5	個体の経験に基づく出力差を用いた慎重な選択	1.0	0.99

式 (7) で示した通り，ε の値は大きければ大きいほど，変更後の活性化関数が関数同士の距離に依存しなくなり，既存手法に近いものになる．これを世代と共に減衰させることで，探索初期では様々な関数へ，探索後期では距離の近い関数へ変更することを期待する．ε_1 を 1.0，減衰率を 0.99 としたとき，300 世代目の $\varepsilon_3 00$ は 0.05 程度になる．

この他に，区間積分差を用いた手法では，どちらの実験でも -1 から $+1$ までの範囲を積分し距離を計算する．個体の経験に基づく出力差を用いた手法では，どちらの実験でもミニバッチサイズは 24 とする．既存手法は現在選択されている関数以外の関数へランダムに変更されるため，ε の値はプログラムに影響しない．

本実験を行うにあたって，すべての実験で共通するパラメータを以下の表に示す．

ひとつの世代には 480 体の個体が存在し，$-2.0, -1.0, -0.5, +0.5, +1.0, +2.0$ のそれぞれの共有重みに対して 4 回の試行をし，劣悪な初期状態による適応度の低下を軽減する．そうして得られた 480 個の適応度のうち，上位 10%である 48 体のネットワーク構造は変更され

表 8: 使用するパラメータ

パラメータ	値
個体数	480
最大世代数	300
変異の際，活性化関数の変更が選択される確率	50%
適応度測定の繰り返し回数	4
選択の際淘汰される確率	10%
選択の際エリート保存される確率	10%

ることなく次世代の個体として保存され，下位 10%である 48 体の個体は淘汰され次世代には受け継がれない．上位 90%である 432 体のネットワークは WANNs に従い，ノードの挿入，シナプスの追加，活性化関数の変更のいずれかの操作を行い次世代の個体となる．このとき活性化関数の変更が変異として選択される確率は 50%である．この操作を 300 世代目まで行い，その時点で最も適応度の高い個体の出力を準最適解として記録する．

6.3　実験結果

謝辞

山口先生有り難う！

参考文献

[1] Zador, Anthony M. ”A critique of pure learning and what artificial neural networks can learn from animal brains.” Nature communications 10.1 (2019): 3770.

[2] Gaier, Adam, and David Ha. ”Weight agnostic neural networks.” Advances in neural information processing systems 32 (2019).

[3] Stanley, Kenneth O., and Risto Miikkulainen. ”Evolving neural networks through augmenting topologies.” Evolutionary computation 10.2 (2002): 99-127.

[4] 坂和正敏, 田中雅博. ”遺伝的アルゴリズム” 朝倉書店 (2002).

[5] Braun, Heinrich, and Joachim Weisbrod. ”Evolving neural feedforward networks.” Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Innsbruck, Austria, 1993. Vienna: Springer Vienna, 1993.

[6] 伊庭斉志. ”進化計算と深層学習 -創発する知能-” オーム社 (2015).

[7] Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri. ”Activation functions in deep learning: A comprehensive survey and benchmark.” Neurocomputing (2022).

[8] 青嶋誠, 矢田和善. ”高次元の統計学” 共立出版 (2019).

[9] Glorot, Xavier, and Yoshua Bengio. ”Understanding the difficulty of training deep feedforward neural networks.” Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010.

[10] Black, James E., et al. "Learning causes synaptogenesis, whereas motor activity causes angiogenesis, in cerebellar cortex of adult rats." Proceedings of the National Academy of Sciences 87.14 (1990): 5568-5572.

[11] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).