The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

アクションゲーム学習におけるモジュラーニューラルネットワークの導入

Introducing Modular Neural Networks in Learning Action Games

高橋　寿徳 *1　　棟朝　雅晴 *2
Toshinori Takahashi　　Masaharu Munetomo

*1北海道大学　工学部
School of Engineering, Hokkaido University

*2北海道大学　情報基盤センター
Information Initiative Center, Hokkaido University

In this paper, we introduce a modular network approach in neuroevolution for action game learning. We employ NEAT (NeuroEvolution of Augmenting Topologies) to generate modular networks learning game stages under different conditions, which are combined to obtain networks that can adapt to difficult situations appeared in actual game playing. We show the effectiveness of our approach in a Pygame instance compared to the original NEAT.

1. はじめに

NEAT(NeuroEvolution of Augmenting Topologies)[1][2] はニューラルネットワークを遺伝的アルゴリズムを用いて進化させ, 入出力に対して最適な接続, 重み付け, バイアス設定を探索する事で問題解決を図る事を目的としており, ニューラルネットワークの集団を構造上の類似性に基づいて種集団に分け, 種集団の中で競争, 交叉を行わせる事で効率的な探索を可能としている.NEAT において, 学習後のネットワークを保存し、それらのネットワークをモジュラーネットワーク [3] として再利用する事で, 単一のニューラルネットワークでは解決が困難な問題に対しても対応できる.

アクションゲーム学習においては同一の規則 (ゲームルール) の下, 同一の状態 (ステージ) から学習を始める事が前提となっており, 同一のステージでも予め他のステージを学習していた場合, 学習の動きがどのように変化するかについて議論は不十分である. 本研究では, 同一のゲームルールを保った状態で, 互いに異なる複数のステージを学習したネットワークを準備し, これらのネットワークを困難なステージの学習に活用した際に,NEAT のみで困難なステージに取り組んだ場合に比べて, モジュラーネットワークの導入により学習結果が改善されるかを調べる事を目的とする.

2. アクションゲームにおける NEAT の適用

今回学習の題材とするのは,Pygame での使用を想定されたゲーム基盤 [4] を用いて実装したアクションゲームであり, ゲームの目的は, 蛇の姿をした自機を操作して, 敵キャラクターや棘といった障害物を乗り越えながらステージ最奥のゴールに触れる事である. 行える操作は, 左移動, ジャンプ, 右移動,(空中でのみ) 急降下, 近接攻撃, 形態変更の 6 つであり, 有意な操作の組み合わせは全 21 通りである.

自機は二形態あり, 形態の違いで性能が変化するが, どちらの形態でもクリアできるものとしている. ゴールに接触すればゲームクリアとなるが, ゲームオーバーとなる条件は,(1) 敵に複数回接触して自機の LIFE が 0 になる,(2) 棘に接触する,(3)TIME が 0 になって時間切れになる, の 3 つである.

NEAT をゲーム学習に適用するために, ゲームから得られる情報を入力, ゲームに対する操作を出力として返すニューラルネットワークのトポロジーを作成する.

入力は表 1 左に記す自機の状態に関する情報と, 自機周囲のオブジェクトの情報を得るために, 表 1 右に基づいて周囲 3 マスの物体の種類を特定する.

表 1: 入力に用いる自機の情報とオブジェクトの情報

自機の情報 ([-1,1] に正規化)	オブジェクトの種類と値
x 方向の速度	何もなし:0.0
y 方向の速度	自機:2.0
現在の形態	ブロック:-1.0
接地しているか	アイテム:-0.5
急降下中か	モンスター:0.5
空中ジャンプできるか	棘:1.0
攻撃できるか	
現在 HP	

また, ネットワークの適合度はステージクリアまで進められたか, 途中でゲームオーバーとなったかで以下の 2 式から求める.

$$fitness = \begin{cases} -10 + \frac{LIFE}{20} + \frac{Score}{50} - \sqrt{|PlayerX - GoalX|} \\ if \quad Gameover \\ 150 + LIFE + \frac{Score}{100} + \frac{restframes}{60} \\ if \quad Stageclear \end{cases} \tag{1}$$

3. モジュラーネットワークの適用

実際のゲームプレイを想定した, 応用ステージに取り組むためのモジュラーネットワークとして, 敵だけが登場するエネミーステージ 15 種類を順不同で学習させたネットワーク 15 種類と, 棘とアイテムだけが登場するジャンプステージ 15 種類を順不同で学習させたネットワーク 15 種類を作成した. 本研究における全体のアルゴリズムは図 1 におけるフローチャートで示される. モジュラーネットワークを作る際の, ゲーム情報を受けて次の操作を決めるネットワークと, モジュラーネット

連絡先: 高橋寿徳, 北海道大学工学部情報エレクトロニクス学科，trpg_igo3576eigo@eis.hokudai.ac.jp

ワークを利用する際の, ゲーム情報を受けて次の 20 フレーム間の操作を担うモジュラーネットワークを決めるネットワークが異なる事に留意する. NEAT のパラメーター設定は,NEAT を用いて各種 Atari ゲームの実験を行う OpenAI-NEAT プロジェクトの設定 [5] を参考にした.

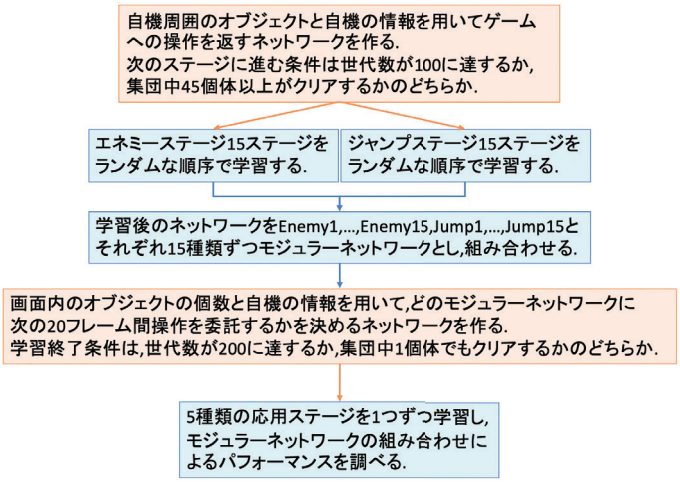

図 1: ゲーム学習におけるモジュラーネットワークを用いた NEAT のフローチャート

応用ステージはエネミーステージとジャンプステージの要素が交互に繋がったステージであり, ステージの長さは各種エネミー, ジャンプステージに比べておよそ 2.5 倍としている. 応用ステージを実践するには, 新たにどのモジュラーネットワークに操作を委託するかを決めるネットワークを作る必要がある. 図 2 左にそのネットワークによって使用するモジュラーネットワークを決定する様子を示し, 図 2 右に作られたモジュラーネットワークが動作する様子を示している.

その出力は, 全部で 31 通りあり, 順に接続なし, エネミーステージネットワーク 15 種, ジャンプステージネットワーク 15 種としており, 接続なしのネットワークは, 入出力が繋がっていないうちにのみ委託される可能性がある (何も操作をしない),

入力は, 自機の情報は表 1 の情報に自機の現在の x 座標と y 座標, 現在スコア, 残り時間を加え, オブジェクト情報は周囲 7 マスの各種オブジェクトの個数についての情報をとる.

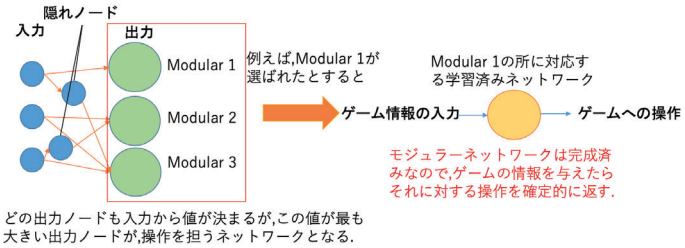

図 2: NEAT によるモジュラーネットワークの決定

また, 応用ステージ実践中の NEAT の出力となる, モジュラーネットワークの切り替えはゲーム内の時間で 20 フレーム毎としており, ゲームへの操作は第二章と同様のゲームの情報をゲームプレイ中常にモジュラーネットワークへの入力として与え, 操作はモジュラーネットワークの出力として確定的に返される.

NEAT のパラメーター設定は, 大半は複数ステージ学習と応用ステージ学習でパラメーター設定は同じだが, 複数ステージ学習では初期ネットワークを無接続状態, 応用ステージ学習ではランダムな入出力 1 組を初期ネットワークとする FS-NEAT と設定している. この理由としては, ランダムな入出力 1 組の構造から始める FS-NEAT の方が, 与えられた一つの応用ステージに対して有効な入出力接続を決め打ちする事で, 効率よく解候補を探索できる動きが見られたためである. 一方で, 複数ステージ学習の場合では, 決め打ちされた入出力が与えられた最初のステージに依存する傾向があり, 第二, 第三のステージを学習させる際に悪影響を与える事が見られたため, 無接続状態から開始している.

応用ステージは 1 ステージずつ学習させる為, 費やす最大の世代数は 200 とし,1 個体でもクリアした個体が出現した時点で学習の 1 サイクルを終了させるものとする.

4. 評価実験

応用ステージは 5 種類用意し, モジュラーネットワークは各々のステージを 15 種類全て学習したネットワーク 30 種類 (FULL) だけではなく, 実験の中で, クリアが容易とされた上位 10 ステージだけを学習させたネットワーク (Easy10) や,15 種類のステージを一つずつのみ学習させたネットワーク (Singles) を用意し, これに第二章と同様に NEAT で応用ステージに取り組んだ結果 (Normal) を加えて互いに比較させた.

今回の実験では, 適合値はスコアやゴールまでの距離など, 複数の要素が絡む上にクリアか否かで大きく値が変わるため, 各ネットワークのステージ毎の評価には,200 世代以内のステージ達成度 (どこまでゴールに近づけたか) の最大を使用した. 達成度が 100 であれば, そのステージをクリアした事を意味する. 各グラフは, 左から順に FULL,Easy10,Singles,NormalNEAT のネットワークを用いて,20 回ステージを学習した際の進行度の統計を箱ひげ図で表しており, グラフ内では応用ステージの事を Multi Stage と呼んでいる. 各応用ステージを 20 回実践した際の達成度について, 特にネットワーク間で有意な差が見られた結果を図 3 から図 7 で示す.

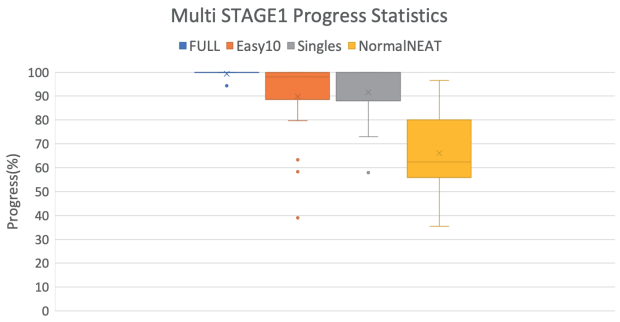

図 3: 応用ステージ 1 を 20 回学習させた時の進行度

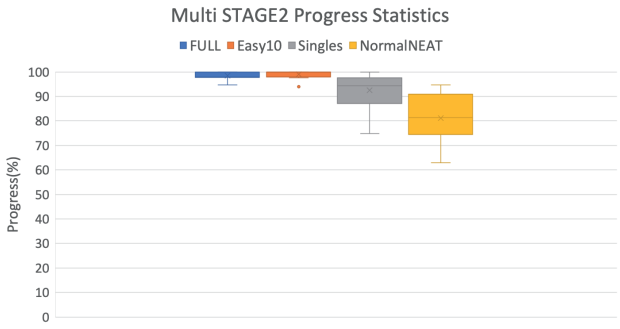

図 4: 応用ステージ 2 を 20 回学習させた時の進行度

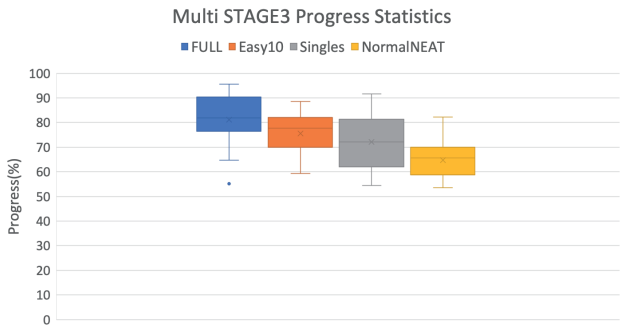

図 5: 応用ステージ 3 を 20 回学習させた時の進行度

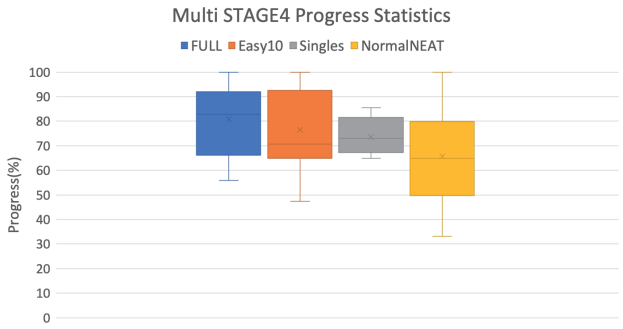

図 6: 応用ステージ 4 を 20 回学習させた時の進行度

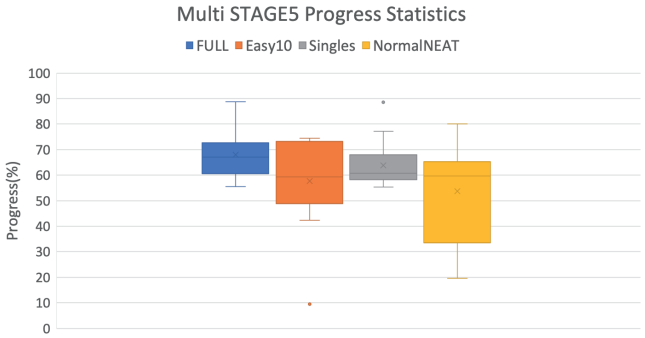

図 7: 応用ステージ 5 を 20 回学習させた時の進行度

この結果から, 応用ステージ 1,2 のように,FULL ネットワークであれば 20 回とも殆どクリアまで近づける難易度であれば,Easy10,Singles といった比較的学習量が少ないネットワークでも, 普通に NEAT で学習させた場合より目に見えて良好なパフォーマンスを出せる事が挙げられる. さらに, 応用ステージ 3 から 5 のように, 殆どクリアが見込めないほどの難易度となると, 多くのステージを学習したネットワークであるほど, 安定して高い進行度を保つことができると言える. 本研究では, いずれの場合でもモジュラーネットワークを通した方が NEAT のみで学習させるより平均的に高い進行度を出せており, 予めステージを学習させた経験を役立てられているといえる.

5. おわりに

本研究では, 敵が登場するステージ, 棘が登場するステージで分けて, 短いステージを複数学習させてネットワークを用意し, そのネットワークを応用ステージに再利用する事で, 直に応用ステージを学習させるよりも効率よくステージ進行を進められる結果を示した.

一方で, 本実験には応用ステージの定義や, モジュラーネットワークを作るために学習させるステージ群は他にも色々な種類が考えられ, 自機の形態なども考慮したステージでネットワークを作るなど, 自機の二形態の特性も含めて学習させられる可能性があるという点で, 多くの検証が行える余地がある.

参考文献

[1] Kenneth O. Stanley and Risto Miikkulainen. ”Efficient Evolution of Neural Network Topologies.” *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600).*

[2] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks through Augmenting Topologies.” *Evolutionary Computation 10(2): 99-127. 2002.*

[3] Bart L.M. Happel and Jacob M.J.Murre. ”The Design and Evolution of Modular Neural Netowork Architectures” *Neural Networks,1994,7,985-1004.*

[4] https://github.com/aidiary/pygame/tree/master/action

[5] https://github.com/HackerShackOfficial/OpenAI-NEAT