卒 業 研 究 報 告 題 目

DQNを利用したリバーシAIに関する研究

A Study on Riversi AI using Deep Q-Networks

指 導 教 員　　出口 利憲 教授

岐 阜 工 業 高 等 専 門 学 校　電気情報工学科

2014E27 長澤 紘汰

平 成 ３ １ 年 (２ ０ １ ９ 年)　　２ 月 １ ５ 日 提 出

Abstract

Recently, the reserch of Artificial Intelligence is applied to various fields of discipline and keep developing rapidly during about 70 years after it was born in Dartmouth conference. Deep Reinforcement Learning is a field which is recently and specially interested in Artifitial Intellingence. The reasons why it can become to decide everything very rapidly and precisely combining the best regression of Deep Learning with the best decision of Reinforcement Learning on a problem having big computational complexity.

In this reserch, we apply the DQN algorithm combined NFQ argorithm which is a one of Deep Reinforcement Learning Algorithms to a game of 8x8 Reversi because we don't have a computer which is embedded Tensor Processer Unit. First, I attempt to choose the best hyper parameters, size of networks and quantity of neurons using its algorithm from random records of Cartpole problem. Second, we apply to 8x8 Reversi and examine the performance of all AI's which are made using Keras in TensorFlow on a construction of Combolutional Neural Networks using Deep Learning. As a result, we found that its AI could initially success to learn 8x8 Reversi strategies using the best hyper parameters with the size of networks and the number of neurons selected through pre-Exp. However, its AI failed in the middle (Table 1). Therefore, in order not to fail to learn, we should make its AI learn itself using pure DQN algorithm.

Table 1　DQN AI's win rate

episode	MTS:100	Random
0	0	
1	0.3	0.8
2	0.5	0.8
3	0.2	1.0
4	0.7	0.9
5	0.3	0.9
6	0.3	0.9
7	0.4	0.9
8	0.4	1.0
9	0.4	0.9

Contents

Chapter 1　Introduciton

人工知能 (AI) という研究分野が、ダートマス会議で誕生してから約 70 年もの間、様々な分野で応用され現在も急速な発展を続けている。応用領域は多岐に渡り、音声・画像認識などの情報学のみならず、医学や経済学など現実世界のあらゆる分野で利用されている。深層強化学習とは、人工知能という研究分野において近年特に注目を浴びている分野である。その主たる要因は、莫大な計算量を持つ問題において、深層学習による非常に優れた回帰と強化学習による優れた最善判断を組み合わせることで、非常に早く最善判断を下すことが可能となったからである。[1)]

本研究では、深層強化学習のアルゴリズムの一つである、DQN を Cartpole 問題に適用し、最善なハイパーパラメータの選定や、最適なネットワークサイズを選定し、ランダム生成した譜面から強化学習を試みる。深層学習面で用いる畳み込みニューラルネットワークの構築には、機械学習向けライブラリである、TensorFlow の Keras を用いる。その後、選定した最適なハイパーパラメータと最適なネットワークサイズを、8x8 のリバーシに適用し、制作した AI すべての性能を比較検討する。

Chapter 2　Neural Network

2.1　Neural Network

ニューラルネットワークとは、人間など生体の脳機能にみられるいくらかの特性に似た数理的モデルであり、シナプスの結合によりネットワークを形成した人工ニューロン (ノード) が、学習によりシナプスの結合強度を変化させて、回帰や分類などの問題解決を図る手法である。[2)]

2.1.1　Neuron

ニューロンとは、Figure 2.1 に示すような生体の神経系を構成する細胞である。人体の脳には、約 1000 億〜2000 億個ものニューロンが存在するといわれており、ニューロンの細胞体一つ一つから軸索と、複雑に枝分かれしている樹状突起が出ていて、それらが別のニューロンと繋がり合って、複雑なネットワークを形成している。

ニューロンには情報伝達の役目がある。ニューロンの情報伝達方法は入力刺激によりニューロンに活動電位が発生し、軸索から終端部へと電位が到達したらその電位によって電位依存性カルシウムイオンチャネルが開くことで、ニューロンの繋ぎ目に存在する構造であるシナプスへと情報が放出され、別のニューロンの樹状突起の受容体へと伝達される。あるニューロンに複数のニューロンから入力したり、入力刺激の強度などによって活動電位の発生閾値も変化させたりすることで、情報の付加が行われたりする。

2.1.2　Neuron Modeling

実際のニューラルネットワークでは、生体ニューロンの情報伝達方法を極めて簡素化した、Figure 2.2 に示すような入力層と出力層のみ存在する人工ニューロンモデルに置き換えて利用する。これは心理学者・計算機学者である Frank Rosenblatt が開発したもので、単純パーセプトロンとも呼ばれる。

このニューロンモデルの出力 y は、式 (2.2) のように計算される。ここで、 x_i は i 番目の入力、 w_i は i 番目の結合荷重、 f は活性化関数である。

$$u = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

$$y = f(u) \tag{2.2}$$

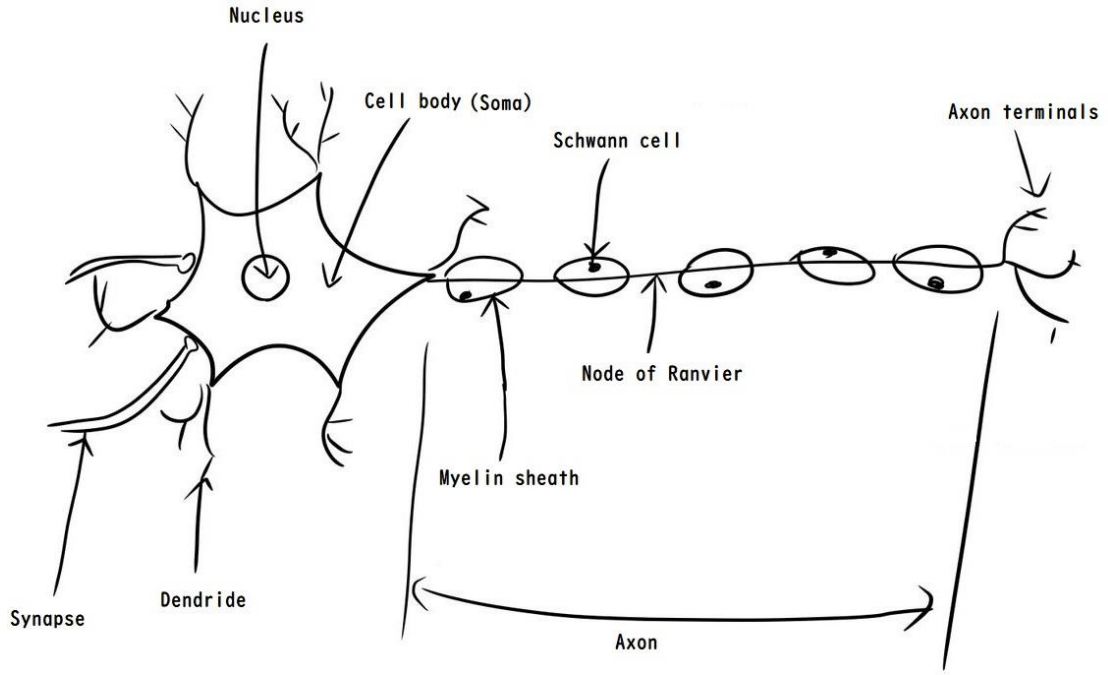

Figure 2.1　Neuron

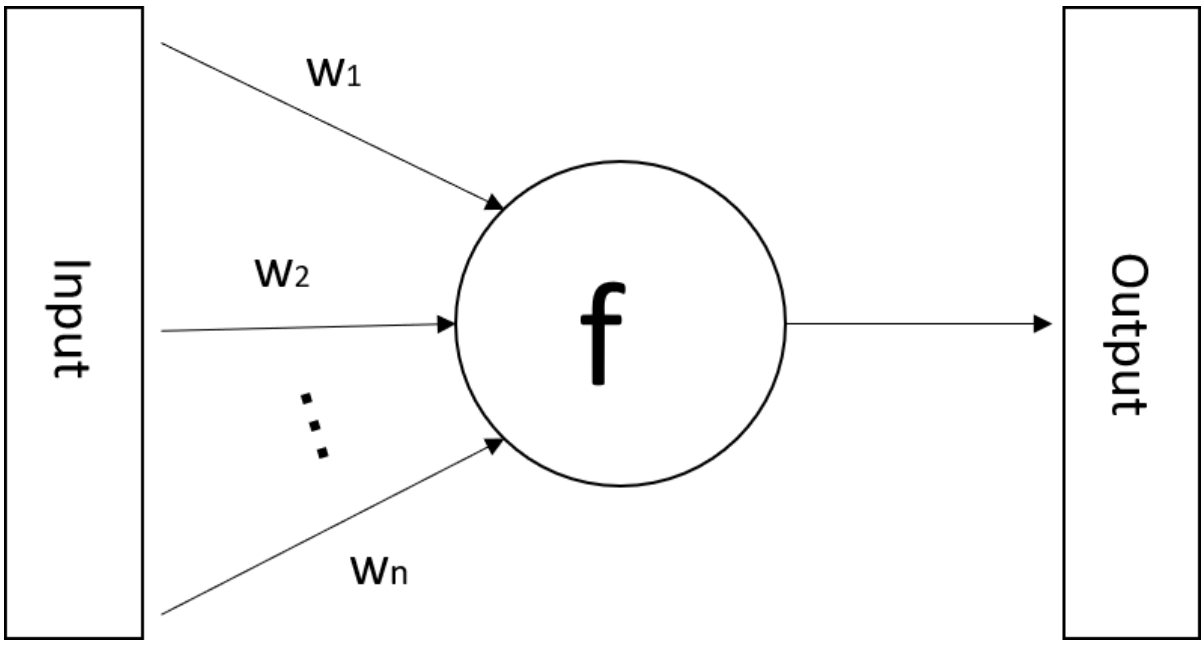

Figure 2.2　Simple Perceptron

2.1.3　Activation Function

活性化関数は伝達関数 (Transfer Function) とも呼ばれ、ニューラルネットワークにおいて線形変換後に適用する非線形関数・恒等関数のことを指す。活性化関数にはいくつかの種類があるが、代表的なものを次に示す。

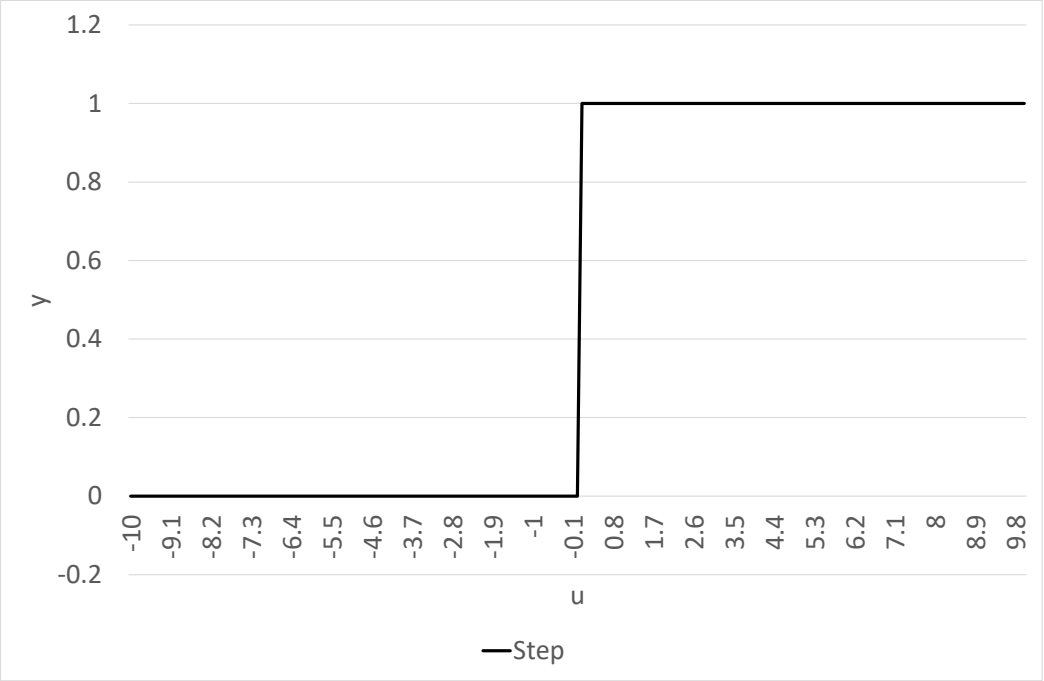

Figure 2.3 Step function

- Step function(Figure 2.3)

$$y = \begin{cases} 1 & (u > 0) \\ 0 & (u \leq 0) \end{cases} \tag{2.3}$$

- Sigmoid function(Figure 2.4)

$$y = \frac{1}{1 + e^{-u}} \tag{2.4}$$

- ReLU function(Figure 2.5)

$$y = \begin{cases} u & (u > 0) \\ 0 & (u \leq 0) \end{cases} \tag{2.5}$$

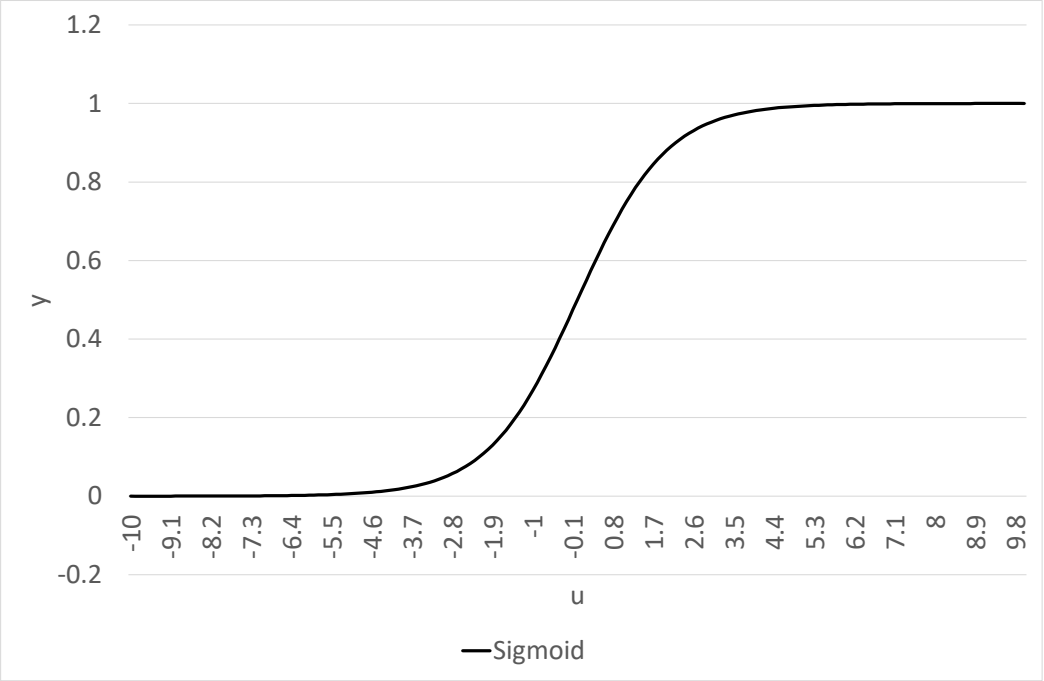

Figure 2.4 Sigmoid function

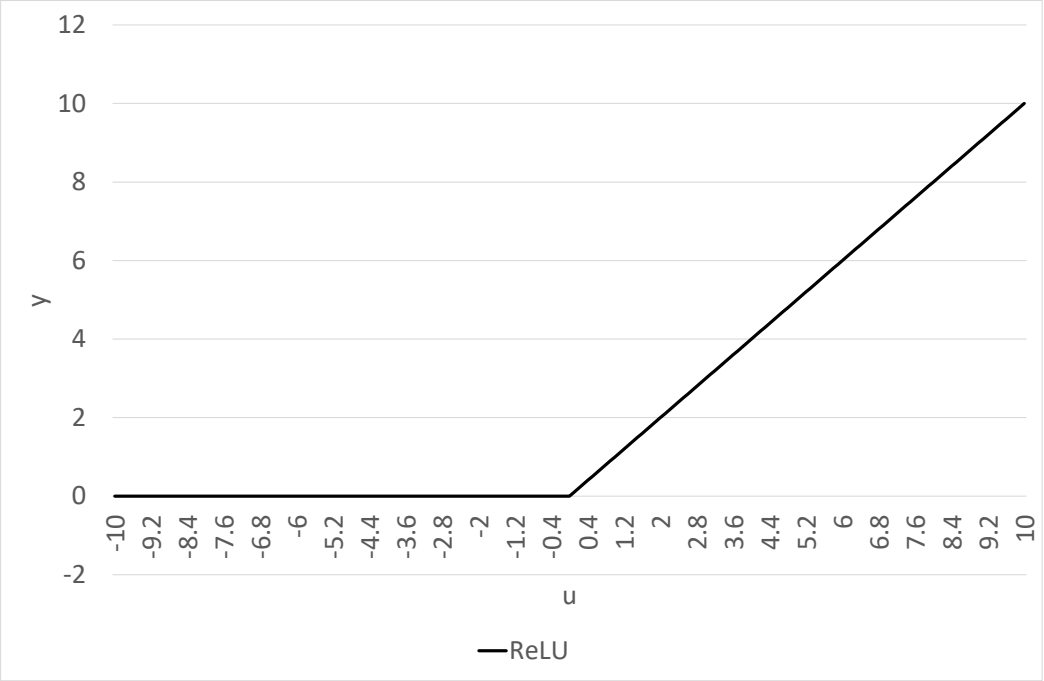

Figure 2.5 ReLU function

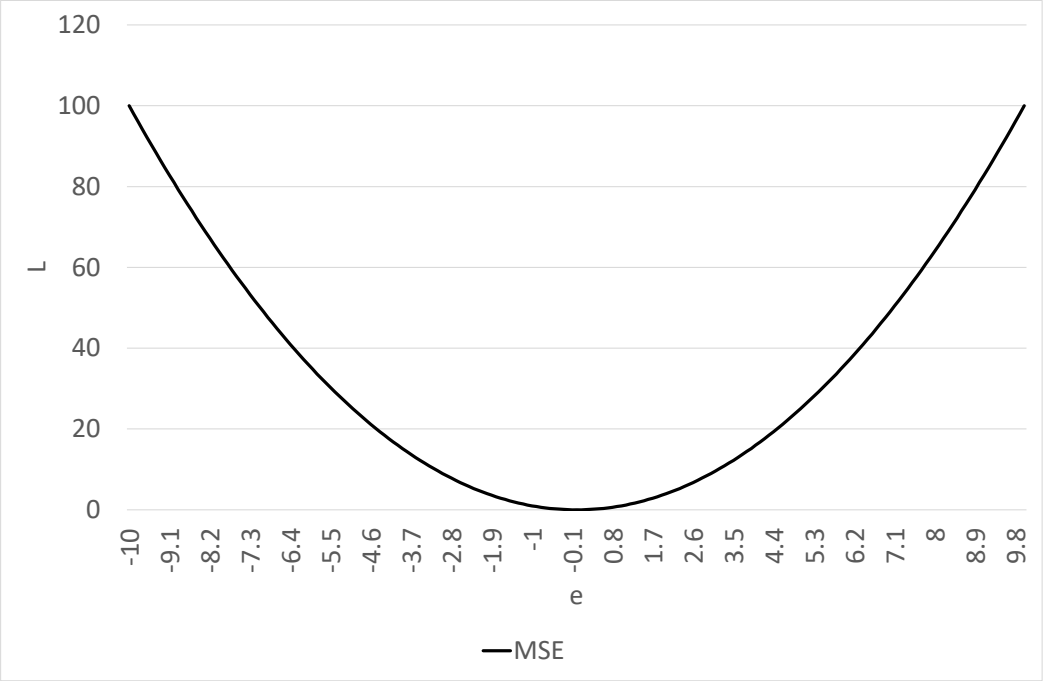

Figure 2.6　Mean Squared Error

2.1.4　Loss Function

損失関数とは、回帰や分類において教師データとの損失 (誤差) を定義する関数である。ニューラルネットワークではこの損失を最小限にする役割があるので、損失関数の選択は、問題や環境によって変わることがある。損失関数として代表的なものを以下に挙げる。

～回帰～

- 平均二乗正規化誤差性能関数 (mse)

  一般的な損失関数である。データが正規分布だと考えられる際の最尤推定 (Figure 2.6)。

  ここで、損失を e 、損失関数を L とする。

- Huber 関数

  学習データの外れ値に影響されにくい損失関数である。主に、ロバスト回帰に用いられる。

- τ-分位損失関数

  最適値から負側に離れているか、正側に離れているかで損失量が変わる損失関数

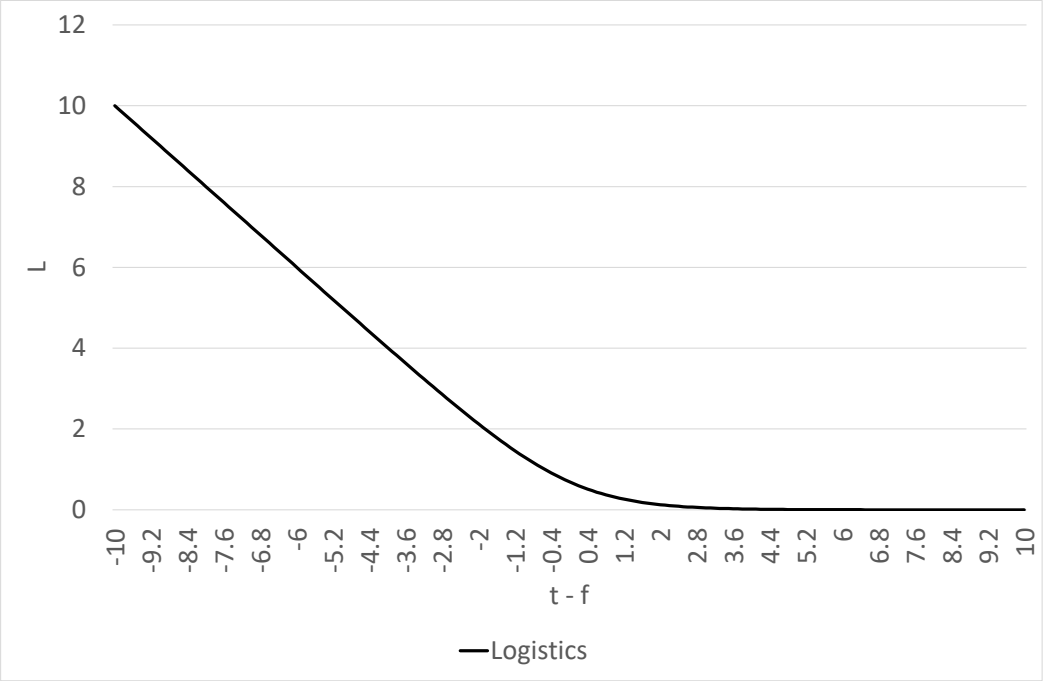

Figure 2.7　Logistics function

である。主に、分位点回帰に用いられる。

- ϵ-感度損失関数

  上手く回帰できていない外れ値に対してのみ重点的に学習を進めていく損失関数である。主に、サポートベクトルマシンに用いられる。

～分類～

- 0-1 損失関数

  分類を間違えた回数を数える損失関数である。単純パーセプトロンの学習性能評価に利用されていた。

- Logisitics 関数

  損失が 0 になることがないが、損失の大きさで分類が成功しているかどうかが判断できる損失関数である。主に、Logistics 回帰で用いられる (Figure 2.7)。

  ここで、損失を $t - f$ 、関数を L とする。

- Hinge 関数

  ある点から負側に損失を与える損失関数である。主に、サポートベクトルマシンで用いられる損失関数。

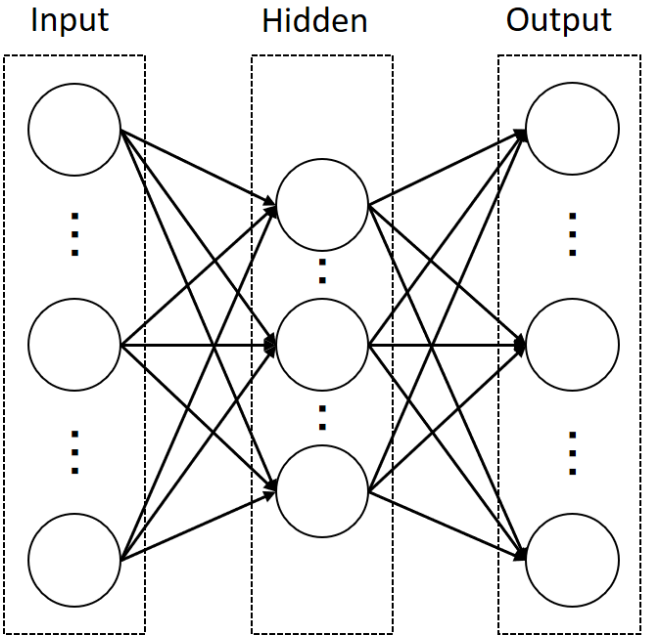

Figure 2.8　Multilayer Perceptron

2.1.5　Multilayer Perceptron

多層パーセプトロンとは Figure 2.8 に示すような、単純パーセプトロンに中間層を 1 層追加した 3 層のノードが存在するパーセプトロンである。このモデルは、認知心理研究者である David E. Rumelhart らにより開発された。また多層パーセプトロンは、単層パーセプトロンでは求解不能であった線形非分離な問題を求解可能である。

2.2　Deep Learning

ディープラーニングとは、中間層を 2 層以上に増やしたニューラルネットワークを用いた機械学習手法のことである。ディープラーニング技術の登場以前では、入力・出力層含む 4 層以上の深層ニューラルネットワークには局所最適解問題や勾配消失問題などの問題が存在していたため、実用化には至らなかった。しかし、コンピュータ科学及び認知心理学研究者である Geoffrey Everest Hinton らの誤差逆伝播法、ボルツマンマシン、オートエンコーダやディープビリーフネットワークを開発した成果により、2010 年代になって初めて深層におけるニューラルネットワークを用いた実用的で効率的な学習が可能となった。ディープニューラルネットワークの代表的なものとして、以下のようなものが存在する。

- 畳み込みニューラルネットワーク

- スタックドオートエンコーダ
- 残差ニューラルネットワーク
- 敵対的生成ネットワーク
- ボルツマンマシン
- 回帰結合型ニューラルネットワーク

2.2.1 Convolutional Neural Network

畳み込みニューラルネットワークとは、順伝播型ディープニューラルネットワークの一種である (Figure 2.9)。 主に画像認識・動画認識・音声認識などの分野で広く用いられており、特に画像認識に関する研究においては、ディープラーニングによる学習手法のほぼ全てが畳み込みニューラルネットワークをベースとしている。このネットワークの各ニューロン間の結合パターンは、生物学における動物の視覚野の配置から来ている。視野の限定された領域における刺激にのみ応答するここの皮質ニューロンは受容野と呼ばれる。

畳み込みニューラルネットワークは、全結合層の他に畳み込み層とプーリング層を組み合わせて構築される。全結合層とは、隣接する層における全てのニューロン間で結合がある層を指す。畳み込み層で行われる処理は、画像処理分野におけるフィルタに相当する。この層で行われる計算は、入力サイズによっても変化するが、かなり大きな計算量となる。プーリング層で行われる処理は、入力画像のサイズを縮小する処理である。この層を通過すると、サイズが約 4 分の 1 程度になるため、何層も重ねられることが多い畳み込み層で増えた計算量に対して、計算負荷の管理に有効な処理である。[2)]

2.3 Learning Method

この世の中に存在するあらゆるニューラルネットワークモデル全てにおいていえることだが、入力される情報から最適な出力を導き出すためには各ノードの最適な結合荷重を学習させる必要がある。これらの学習方法は、最適化手法とも呼ばれている。最も基本的な勾配降下法や確率的勾配降下法、並列分散処理研究などに用いられてきた誤差逆伝搬法や、深層学習のための新たな最適化手法である RMSprop 法や Adam 法が実用化されている他、勾配法によらない最適化手法についても研究されている。[2)]

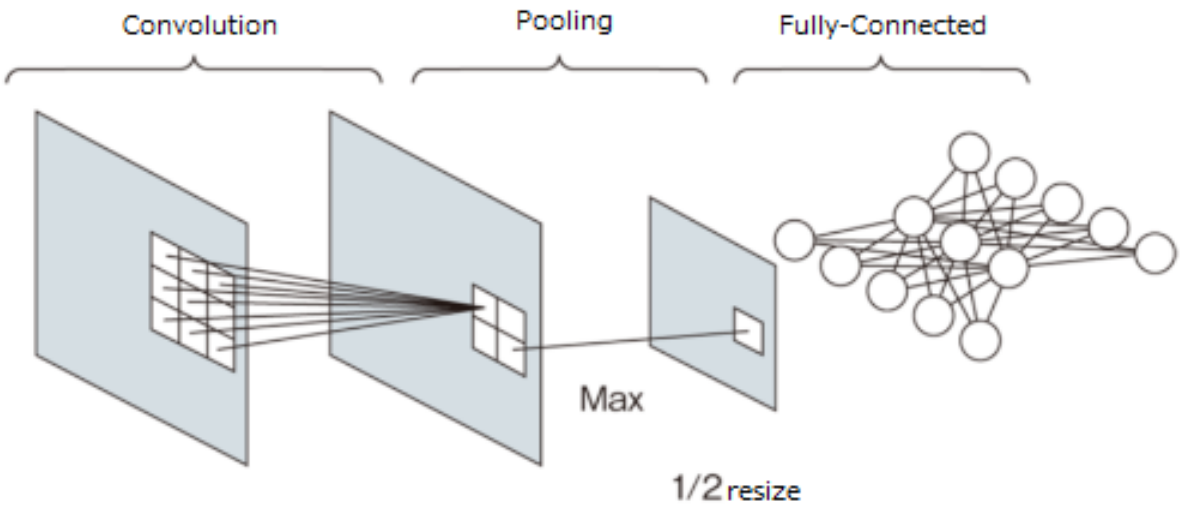

Figure 2.9　Convolutional Neural Network

2.3.1　Optimizer

最適化学習方法としては、勾配法によるものが一般的であるため、勾配法をもとにしたアルゴリズムを以下に示す。

- Stochastic gradient decent (SGD)

  確率的勾配降下法は、最適化処理の中でも初期に提唱された最も基本的なアルゴリズムである。式 (2.6) で重み $\mathbf{w}$ の更新を行っている。

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^{t} - \eta \frac{\partial L(\mathbf{w}^{t})}{\partial \mathbf{w}^{t}} \tag{2.6}$$

  ここで t を世代、 η を学習率、 L を損失関数とする。

- Backpropagation

  バックプロパゲーションとは誤差逆伝搬法とも呼ばれ、3 層以上のニューラルネットワークを学習させるために、1986 年に認知心理研究者であり、多層パーセプトロンの開発者でもある David E. Rumelhart らが考案した機械学習の手法である。適用可能である条件を以下に示す。

  – 人工ニューロンで使われる活性化関数が可微分でなければならない。
  – 勾配が 0 に近い領域が存在する関数を活性化関数に用いることは避けるべき

である。

バックプロパゲーションによる学習は、結合荷重 W_{kj} を出力層側から修正していくため、後ろ向き演算と呼ばれる。出力層における 1 回の誤差逆伝播で変化する結合荷重 ΔW_{kj} は式 (2.7) で表される。

$$\begin{aligned}\Delta W_{kj} &= -\eta \frac{\partial}{\partial W_{kj}} \frac{1}{2} \sum_{i=1}^{n} (t_i - O_i)^2 \\ &= \eta (t_k - O_k) O_k (1 - O_k) H_j \end{aligned} \tag{2.7}$$

ここで、 O_k は出力層の k 番目のニューロンの出力値、 t_k はその教師信号データ (正解データ) であり、 H_j は、中間層の j 番目のニューロンの出力値、 η は人間が設定しなければならないハイパーパラメータの一つである、学習係数である。この係数は、大きすぎると安定性が下がり、小さすぎると学習時間が増大してしまうため、適切に選択する必要がある。この値は AdaGrad 法を組み合わせれば、自動設定される。同じく中間層の微小結合荷重 ΔW_{ji} は、以下の式 (2.8) のようになる。

$$\begin{aligned}\Delta W_{ji} &= -\eta \frac{\partial}{\partial W_{ji}} \frac{1}{2} \sum_{i=1}^{n} (t_i - O_i)^2 \\ &= \eta H_j (1 - H_j) \sum_{k=1}^{n} \{ W_{kj} (t_k - O_k) O_k (1 - O_k) \} X_i \end{aligned} \tag{2.8}$$

ここで、 X_i は入力層の i 番目のニューロンの入力値である。これらすべての結合荷重を計算することで、勾配法によって最適な値に収束させることができる。

- AdaGrad

  AdaGrad とは、2011 年に統計・電気工学研究者である John C. Duchi らが提唱した手法である。SGD と違い、こちらは初期学習率を設定すれば、それ以降の学習率を自動調整可能である。式 (2.12) で重み $\mathbf{w}$ の更新を行っている。

$$h_0 = \epsilon \tag{2.9}$$

$$h_t = h_{t-1} + \nabla L(\mathbf{w}^t)^2 \tag{2.10}$$

$$\eta_t = \frac{\eta_0}{\sqrt{h_t}} \tag{2.11}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \nabla E(\mathbf{w}^t) \tag{2.12}$$

  ここで、 ϵ を初期勾配、 η_0 を初期学習率とする。

- RMSprop

RMSprop とは、2012 年に機械学習研究者である Tijmen Tieleman らが提唱した手法である。AdaGrad を改良したアルゴリズムであり、勾配の二乗指数平均を取るように変更されたものである。式 (2.15) で重み $\mathbf{w}$ の更新を行っている。

$$h_t = \alpha h_{t-1} + (1-\alpha)\nabla L(\mathbf{w}^t)^2 \tag{2.13}$$

$$\eta_t = \frac{\eta_0}{\sqrt{h_t} + \epsilon} \tag{2.14}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \nabla L(\mathbf{w}^t) \tag{2.15}$$

ここで、 α を割引率とする。

- Adam

  Adam とは、2012 年にコンピュータ科学者であり、Google Brain team に所属する Diederik P. Kingma らが提唱した手法である。以上三点の手法を改良し、Momentam 法の概念を取り入れたアルゴリズムである。AdaGrad における初期学習率は、Adam においては存在しない。式 (2.20) で重み $\mathbf{w}$ の更新を行っている。

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla L(\mathbf{w}^t) \tag{2.16}$$

$$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla L(\mathbf{w}^t)^2 \tag{2.17}$$

$$\hat{m} = \frac{m_{t+1}}{1-\beta_1^t} \tag{2.18}$$

$$\hat{v} = \frac{v_{t+1}}{1-\beta_2^t} \tag{2.19}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \tag{2.20}$$

ここで、 β_1 を m 勾配割引率、 β_2 を v 勾配割引率、 α を重みの学習率とする。これまでの手法で移動指数平均を用いた際には、バイアスが変化してしまうという欠点があったが、この手法では $\hat{m}$ と $\hat{v}$ の比率によってバイアスが等倍となり、欠点を解消しているという特徴がある。

環境によっては、Adam より AdaGrad が高い精度を出す場合も以前の研究などで明らかになっているため、使う環境ごとに最適化処理も最適なものを選択すべきであるといえる。

Chapter 3　Reinforcement Learning

強化学習とは、ある環境におけるエージェントが、現在の状態を観測し、取るべき行動を決定する問題を扱う機械学習の一種である (Figure 3.1)。 エージェントは行動を選択することでその環境から報酬を得ることになる。この学習においては、それら一連の行動を通して報酬が最も多く得られるような方策を学習するものである。この学習のアルゴリズムは、動的計画法に類似している。主な手法としてモンテカルロ法、TD 学習や Q 学習がある。[3)]

強化学習における基本的な環境とは、有限状態数のマルコフ決定過程 (MDP) として定式化される。マルコフ決定過程とは、過程における将来の条件付き確率が過去のいかなる状態にも影響を受けない性質であるマルコフ性を満たした、状態遷移が生じる動的システムの確率モデルである (Figure 3.2)。 マルコフ決定過程における学習過程は以下の条件が課される。

- 環境は状態を持ち、その状態は完全で正確に観測可能であること。
- 遷移確率と報酬が得られる確率は事前には与えられず、学習過程で学習していくこと。
- 報酬の指数移動平均を最大化するように行動すること。

強化学習における環境が完全・正確に観測可能ではない場合においては、部分観測マルコフ決定過程 (POMDP) と呼ばれ、belief MDP などの手法によってマルコフ決定過程と同様の手法が適用できるようになる。

3.1　Q-Learning

Q 学習とは、コンピュータ科学者である Richard S. Sutton が 1984 年に提案した TD(λ) 学習を同じくコンピュータ科学者である Christopher J.C.H. Watkins らが 1989 年に方策オフ型に発展させた手法である (Figure 3.3)。[3)] この学習は、有限マルコフ決定過程においてすべての状態が十分にサンプリングできるようなエピソードを無限回試行した場合、最適な評価値に収束することが理論的に証明されている。この学習ではその状態において取れる行動 π の中で、行動全てに対し長期的な価値、つまり有効性を示す Q 値という値を持たせ、エージェントが行動するたびにその値を更新する。Q 値は状態行動価値とも呼ばれる。例えばエージェントの現在の状態を s_t とし、この状態で可能な行動

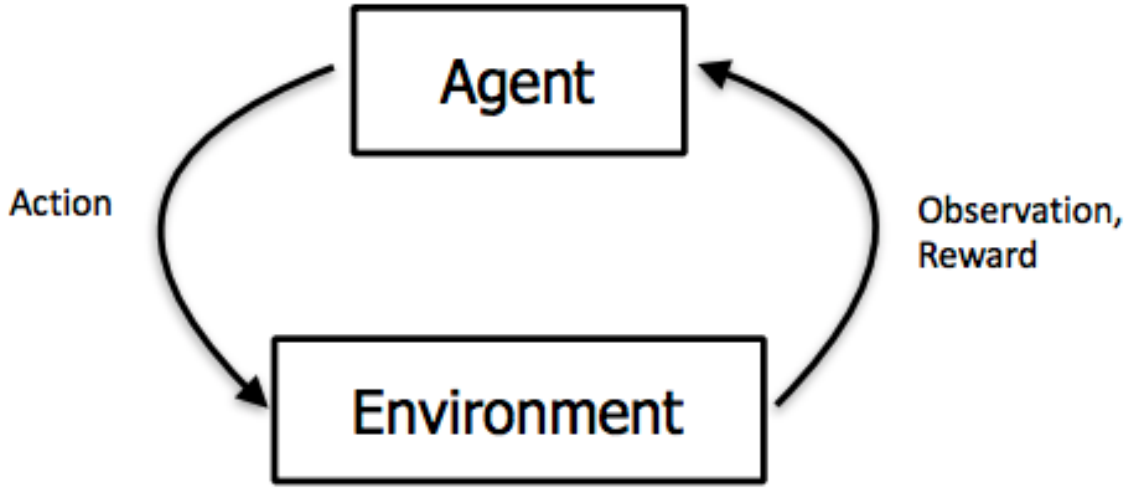

Figure 3.1　Reinforcement Learning

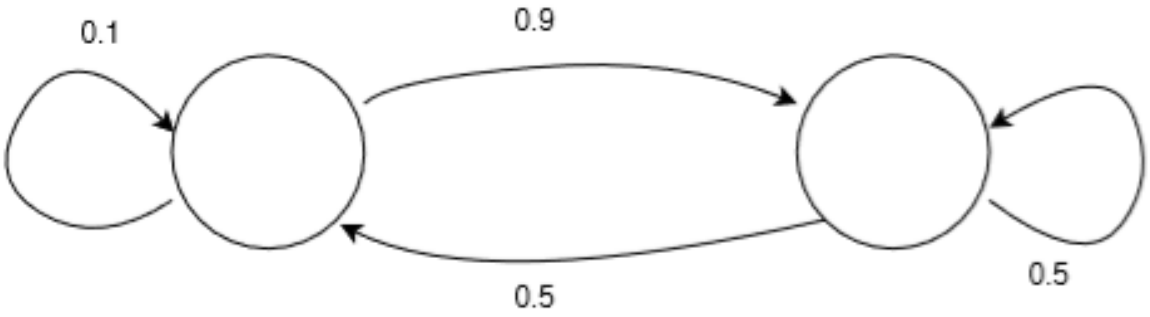

Figure 3.2　Markov decision process

が a 、 b 、 c 、 d の 4 通りあるとする。

この時エージェントは 4 つの Q 値、 $Q(s_t, a)$ 、$Q(s_t, b)$ 、 $Q(s_t, c)$ 、 $Q(s_t, d)$ を元に、次に行う行動を決定する。行動の決定方法は理論上では無限回数試行するならランダムでも Q 値の収束は証明されているが、収束を早めるため、なるべく Q 値の大きな行動が

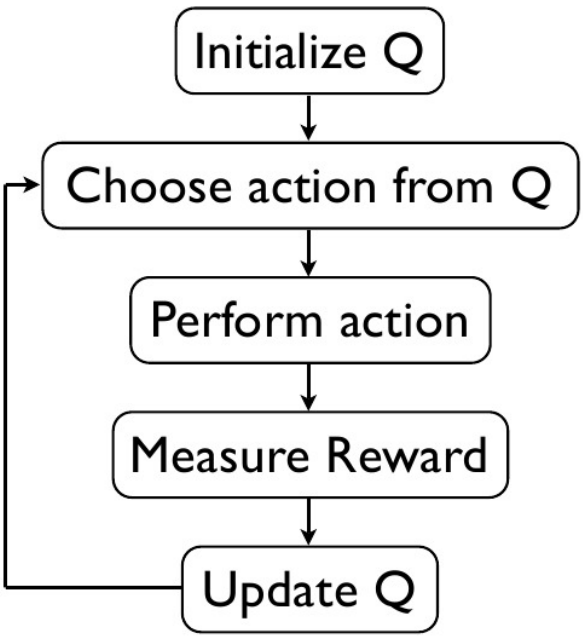

Figure 3.3　Q-Learning

高確率で選ばれるように行う。選択方法としては、ある小さな確率 ϵ でランダムに選択し、それ以外では Q 値の最大の行動を選択する ϵ -greedy 手法や、遺伝的アルゴリズムで使用されているルーレット選択、式 (3.1) のようなボルツマン分布を利用した softmax 手法などが使用されている。

$$\pi(s,a) = \frac{\exp(\frac{Q(s,a)}{T})}{\sum_{p \in A} \exp(\frac{Q(s,p)}{T})} \tag{3.1}$$

ここで T は正の定数、 A は状態 s でエージェントが可能な行動の集合である。

行動を決定した場合、次にその状態と行動の Q 値を更新する。例として状態 s_t のエージェントが行動 a を選び、状態が s_{t+1} に遷移したとするときは、 $Q(s_t, a)$ を式 (3.2) で更新する。

$$Q(s_t,a) \leftarrow Q(s_t,a) + \alpha[r_{t+1} + \gamma \max_p Q(s_{t+1},a) - Q(s_t,a)] \tag{3.2}$$

ここで α は学習率といい後述する条件を満たすパラメータであり、 γ は割引率といい 0 以上 1 以下のパラメータである。また r_{t+1} はエージェントが s_{t+1} に遷移したときに得た報酬である。式 (3.2) は現在の状態から次の状態に移ったとき、その Q 値を次の状態で最も Q 値の高い状態の値に近づけることを意味している。このことにより、ある状態で高い報酬を得た場合はその状態に到達することが可能な状態にもその報酬が更新ごと

Table 3.1　Q-Table

State\Action	a_1	a_2	a_3	a_n
s_0	0	0.24	0.59	0.77
s_1	-0.14	0.02	-0.57	0.24
s_2	-0.85	-0.04	-0.38	-0.95
s_m	0.07	0.15	1.0	-1.0

に伝播することになる。これにより、最適な状態遷移の学習が行われる。 α や γ などのパラメータは、ハイパーパラメータである。この学習は学習率 α が式 (3.3) 、 式 (3.4) の条件を満たすとき、全ての Q 値は必ず最適な値に収束することが証明されている。

$$\sum_{t=0}^{\infty} \alpha(t) \to \infty \tag{3.3}$$

$$\sum_{t=0}^{\infty} \alpha(t)^2 < \infty \tag{3.4}$$

こういった収束性の良さはこの学習の利点ではあるが、以下のようないくつかの問題点も存在する。

- Q 学習による理論的保証は値の収束性のみであり、収束途中の値には具体的な合理性が認められないこと
- ハイパーパラメータの変化に敏感でありその調整に多くの手間が必要であること
- 行動パターンの多い環境において、最適な Q 値の算出にかなりの時間がかかること

行動パターンの多い環境において、行動に対する Q 値をまとめた Q 値テーブル (Table 3.1) を作ることがほぼ不可能であるため、しばしば用いられる手法として関数近似のためにニューラルネットワークを用いた Neural Fitted Q Iteration やその手法を深層化した Deep Q-Networks というものが存在する。

3.1.1　ϵ -greedy Algorithm

ϵ -greedy 手法とは、確率 ϵ でランダムな値を選択し、確率 ($1-\epsilon$) で学習経過上の最適値を選択するという手法である (Figure 3.4) 。通常の学習では、ϵ 値は強化学習の学習

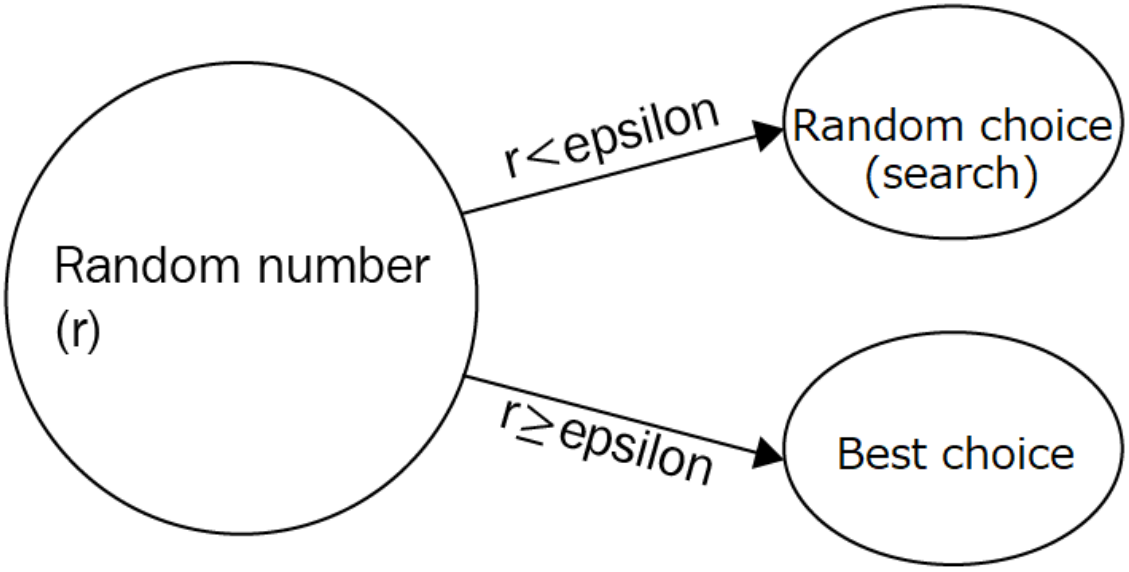

Figure 3.4　ϵ -greedy Algorithm

世代ごとに小さくなるように変化させるため、学習初期にはランダム選択で最適値を探索させることができ、学習終盤は高確率で真の最適値が選択され続けるようになる。この手法では、学習初期段階における仮の最適値が常に選択され続けるような問題や、学習中盤での局所最適解問題を回避することが可能になる。

3.2　Neural Fitted Q Iteration (NFQ)

NFQ は、Q 学習の Q 値を多層パーセプトロンを用いて近似する手法の一つで、学習中にはデータを追加せず、事前に集められたデータのみから学習を行う手法である。この手法は、コンピュータ科学者である Martin Riedmiller によって 2005 年に提案された。この手法の特徴は、重みの更新を行わずに行動と報酬の履歴を増やしながら、ニューラルネットワークの学習を行う。これにより、Q 値の更新が飛び飛びの頻度で行われている。この特徴は、NFQ におけるエージェントの方策が短い周期で激しく変動するリスクを回避していることになる。[4)]

3.3　Deep Q-Networks (DQN)

DQN は、Q 学習と深層学習を組み合わせて Q 値を近似する手法であり、英 DeepMind 社所属のコンピュータ科学者である Volodymyr Mnih や David Silver らによって 2013 年

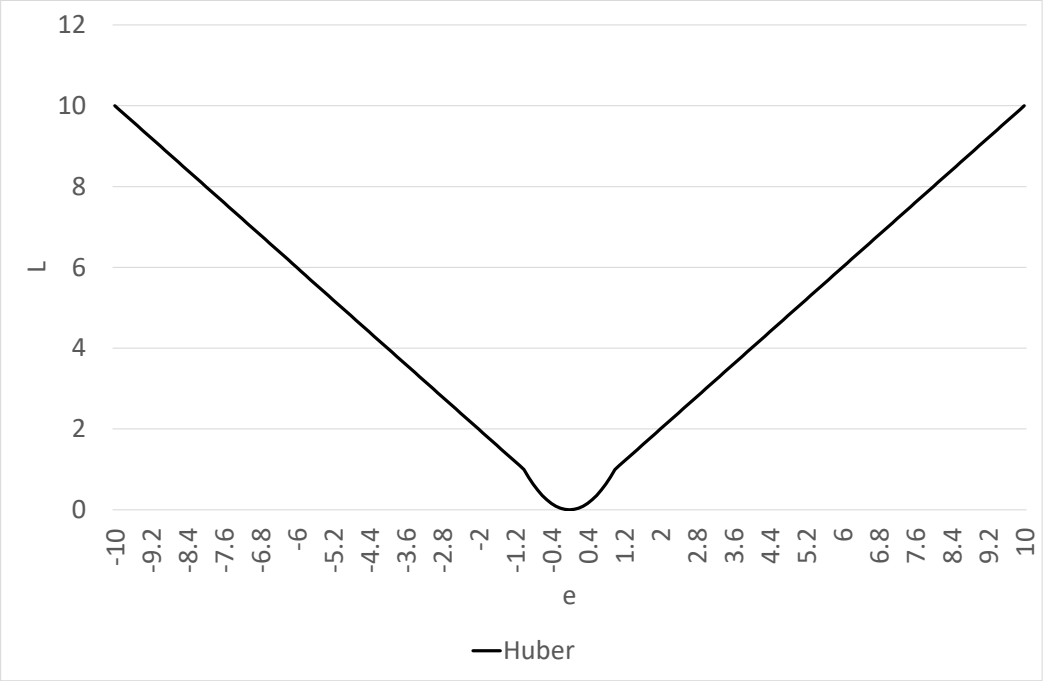

Figure 3.5　Huber function

に提案された。深層学習面では畳み込みニューラルネットワークが利用される。[5] ニューラルネットワークの教師データは、Q 学習の Q 値更新式における差分の考え方により、式 (3.5) で表される。

$$R(s,a) + \gamma \max_{a' \in A} Q(s', a') \tag{3.5}$$

また、誤差 e_{θ_i} と誤差関数 L_{θ_i} は、式 (3.6)、式 (3.7) で表される。

$$e_{\theta_i} = R(s,a) + \gamma \max_{a' \in A} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i) \tag{3.6}$$

$$L_{\theta_i} = \begin{cases} E\left[\dfrac{1}{2}(e_{\theta_i})^2\right] & (|e_{\theta_i}| \leq 1) \\ |e_{\theta_i}| & (|e_{\theta_i}| > 1) \end{cases} \tag{3.7}$$

ここで、 θ_i は第 i 世代のニューラルネットワークの重みである。誤差関数は、通常使用される二乗誤差 E ではなく、Huber 関数を利用する (Figure 3.5)。誤差が -1 〜 1 の間は通常の二乗誤差であるが、それ以外の値では、誤差の絶対値を返す関数である。二乗誤差では誤差が大きい場合において出力が大きくなり、学習が安定しづらい問題を Huber 関数によって解消する事ができる。

勾配 $\nabla L(\theta_i)$ は、式 (3.8) となる。

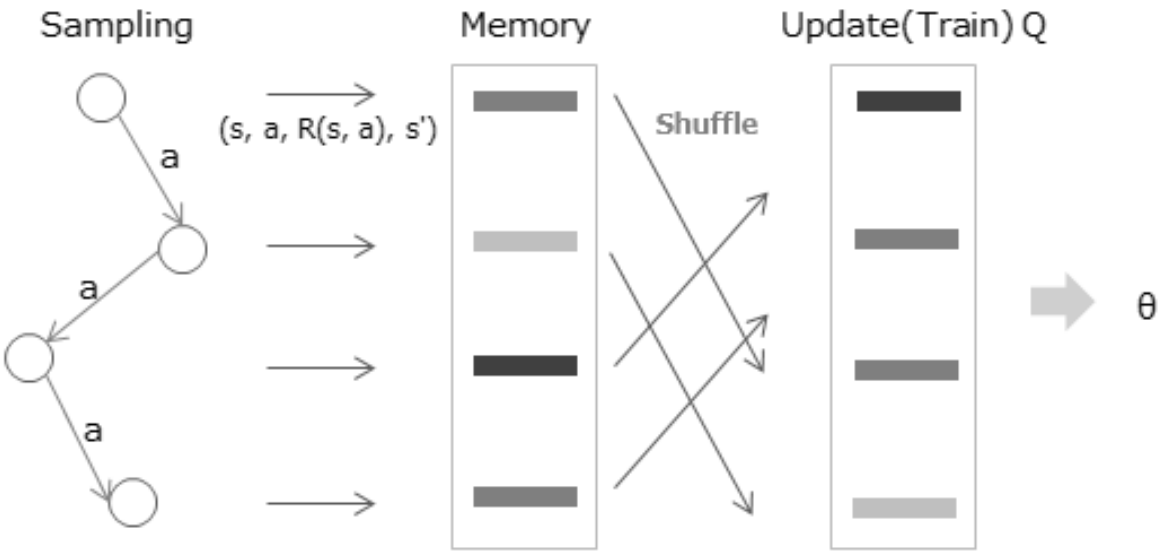

Figure 3.6　Experience Replay

$$\nabla L(\theta_i) = E\Big[(R(s,a) + \gamma \max_{a' \in A} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i))\nabla Q(s,a;\theta_i)\Big] \qquad (3.8)$$

ここで、 $E[A]$ は A の期待値を表し、 θ は、各ノードの重みを表す。その他の文字はすべてQ学習の式と同様である。強化学習面では、 ϵ -greedy 手法を用いることが多い。しかし入力データの学習順や、環境のマルコフ決定過程へのモデル化などには制限があり、Experience Replay や Fixed Target Q-Network、報酬の clipping という手法を用いて工夫する必要がある。[6)]

3.3.1　Experience Replay

Experience Replay とは、時系列順に並んでいる入力データを順番に利用して学習を進めていくと、入力データ間の時系列における相関が影響して過学習を引き起こす可能性があるため、メモリにある程度入力データを保存しておいて、メモリが全て埋まったらランダムにデータを取り出して学習を行う手法である (Figure 3.6)。

3.3.2　Fixed Target Q-Network

Fixed Target Q-Network とは、DQN におけるニューラルネットワークの教師データの重みが一回の学習ごとに毎回更新していると、教師データが常に変化してしまうことから、収束が不安定になってしまう。そのため、一定期間重みを固定して、収束を安定

化させる手法である。

3.3.3 Reward Clipping

報酬の Clipping とは、各ステップ・各ゲームごとで得られる報酬を -1 、0 、1 のいずれかに固定しておく手法である。報酬が固定されていることで、環境によってハイパーパラメータを変化させなくても、深層学習を実行しやすいという利点があるため、この手法を用いることが多い。ゲームにおいては、負けを -1 、引き分けを 0 、勝ちを 1 と設定することが多い。

Chapter 4　Experiment

今回の実験ではまずDQNの特性である報酬のclippingの効果を利用して、Cartpole問題においてDQNにおける最適なハイパーパラメータと、ニューラルネットの層数・ニューロン数を選定する事前実験を行う。事前実験で最適値を選定した後、本実験である8x8リバーシへとその値を利用したDQNを実装する。今回はすべての実験において、盤面を縮小しぼかし処理をする必要がないため、演算量は増加するが畳み込みニューラルネットワークのプーリング層は存在せず、畳み込み層のみ存在するネットワークで実験を行う。なお、活性化関数はすべてReLU関数、最適化手法はすべてAdam法、損失関数はすべてHuber関数とする。

4.1　Environment Setting

学習における環境設定を以下に示す。

- CPU: Intel Core i7 5930K
- GPU: Nvidia Geforce GTX 980Ti
- GPUメモリ: 6GBの内、上限3割を学習に割り当て
- OS: Windows10 Pro
- ニューラルネット作成: Python TensorflowのKeras[7)8)]

4.2　pre-Exp

事前実験では、pythonのGymパッケージに存在するCartPole-v0という実行環境を利用する。CartPole-v0は、ミリ秒単位でカートを右に動かすか左に動かすかを選択するステップを一定回数繰り返すモデルである(Figure 4.1)。 各ステップごとにカート上に立つ棒が倒れなければ報酬+1で、倒れたら報酬0を獲得できる。ニューラルネットワークの入力層は、基本的にカートの位置、カートの並進速度、棒の角度、棒の角速度の4入力とし、出力層は、カートを動かす方向(右or左)のQ値である2出力とする(Figure 4.2)。結果はCSVファイルに、各episodeの累積報酬平均を出力し、グラフ化することで考察する。そのうえで最適値の選定も行う。この環境における初期設定は以下の通りである。

- episode数上限:1000episodes
- episodeあたりのstep数: 200steps

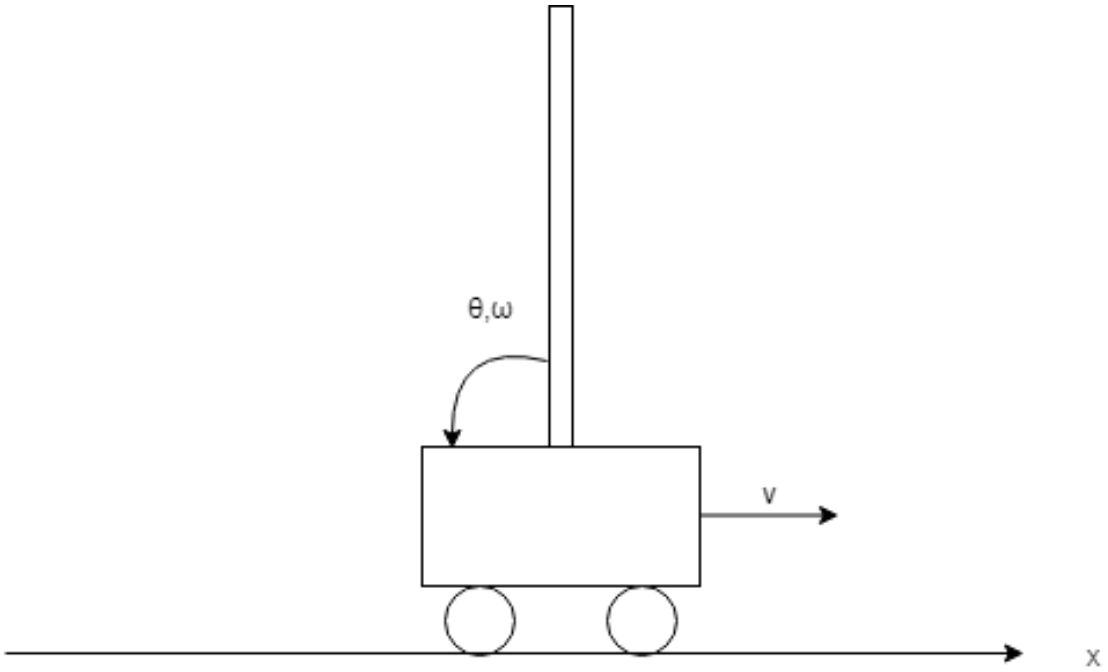

Figure 4.1　CartPole-v0

- 累積報酬の平均計算を行う試行回数: 10steps
- 学習終了条件の累積報酬平均: 195rewards 以上
- Experience Replay 用メモリサイズ: 10000
- Fixed Target Q-Network 用バッチサイズ: 32

これらの条件は、事前実験で変化させることのない条件である。なおメモリサイズとバッチサイズは、DQN におけるハイパーパラメータに該当する。

4.2.1　pre-Exp1: Hyper Parametars

最適なハイパーパラメータの選定では変化させるハイパーパラメータを、Q 学習における学習率 α と、割引率 γ とし、収束速度のみにおいて選定する。ネットワーク構成は以下に示す。

- 入力層: ニューロン 4 個 (位置, 速度, 角度, 角速度)
- 隠れ層 1: ニューロン 16 個 (畳み込み)
- 隠れ層 2: ニューロン 16 個 (畳み込み)
- 出力層: ニューロン 2 個 (右 Q 値, 左 Q 値)

学習率は、 0.00001 、 0.0001 、 0.001 、 0.01 を採用し、割引率は、 0.99 、 0.97 、 0.95 を採用して実験を行う。結果はこれらを互いに組み合わせた 12 のグラフから判断

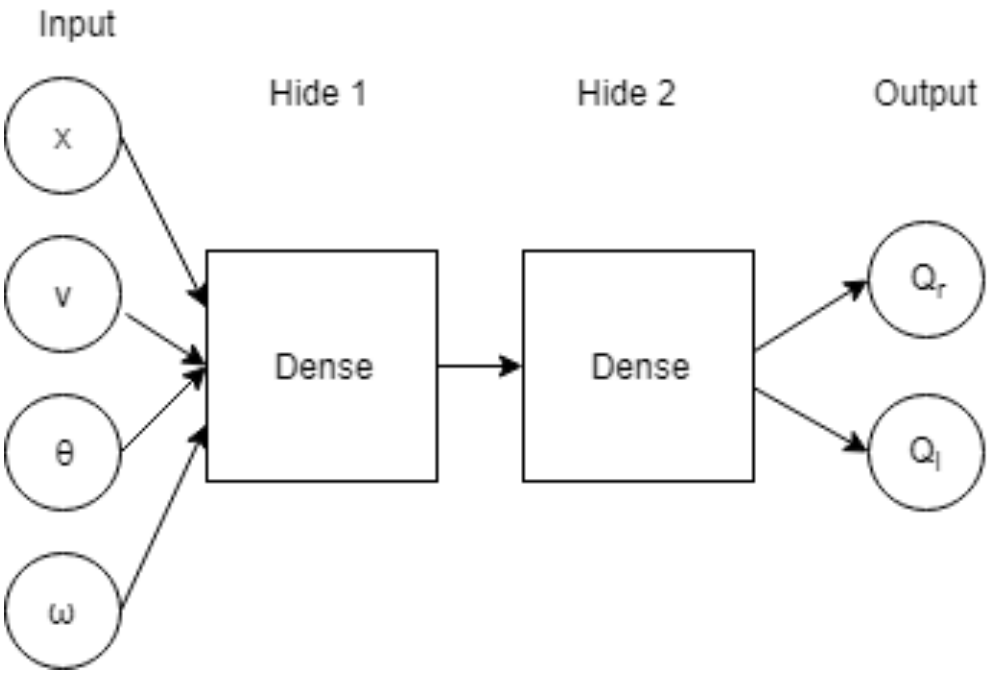

Figure 4.2　A network for CartPole-v0

する。

4.2.2　pre-Exp2: Networks

最適なネットワークサイズの選定では、グラフによる収束過程の特徴により選定を行う。選定が速さではない理由として、本実験である 8x8 リバーシの入力は、CartPole 問題の入力に比べニューロン数が、16 倍の量であるため、単純な速さ比較が最適であるとは考えにくいからである。Q 学習における学習率 α と、割引率 γ はそれぞれ、 0.001 、 0.95 で固定し、ニューロン数は隠れ層すべてにおいて 16 個として実験を行う。ネットワークサイズは、3 層、4 層、5 層を採用する。プログラム上では、ネットワークの大きさが変化しているか確認するため、Tensorflow の Keras 内の plot_model によって、ネットワーク構造の可視化するための画像出力を行っている (Figure 4.3)。

4.2.3　pre-Exp3: Neurons

最適なニューロン数の選定では、グラフによる収束過程の特徴により選定を行う。こちらも同じく、Q 学習における学習率 α と、割引率 γ はそれぞれ、 0.001 、 0.95 で固定し、ネットワークサイズは 4 層で固定する。

ニューロン数は、 (隠れ層 1 ニューロン数, 隠れ層 2 ニューロン数) とすると、$(8, 8)$ 、

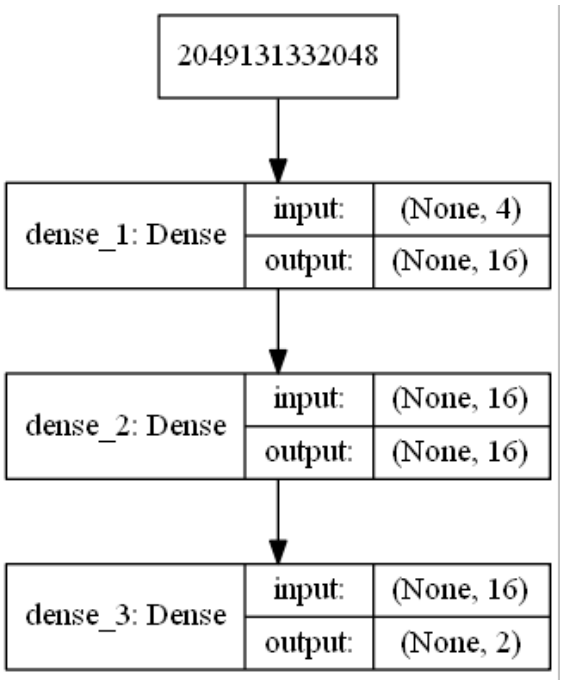

Figure 4.3　Output Image

$(8, 16)$ 、 $(16, 8)$ 、 $(16, 16)$ 、 $(16, 32)$ 、 $(32, 16)$ 、 $(32, 32)$ の 7 パターンを採用して、実験を行う。

4.3　Experiment:Reversi AI

8x8 リバーシへの実装では、事前実験で選定したハイパーパラメータをすべて適用し、DQN のネットワークデータを世代ごとに保存してそれぞれの AI を、事前に作成した可変探索深度 100 のモンテカルロ木探索アルゴリズム (MTS:100) を用いた AI と対戦し、勝率を比較・検討する。またランダム AI とも対戦を行い、勝率を比較・検討する。本実験で利用する AI のアルゴリズムは演算量の都合上、DQN のみでは時間がかかりすぎる。そのため、ニューラルネットの学習においては事前に約 26 万譜面用意し、それをもとに NFQ を用いてニューラルネットの重みを学習させる。そして DQN は、事前譜面を学習させた後に、自己 AI と対戦させニューラルネットワークの最善モデルを選択する際に利用する。つまり 500 譜面を学習させる毎に、AI の性能を MTS:50 と対戦させ、勝率が優れている方を選択・保存する。また 26 万譜面すべて学習し終えたら、学習済み AI を保存し世代を更新する。実時間で 1 週間ほど学習させたら、学習を打ち切り各世代ごとで性能評価する。ここでリバーシ AI の工夫として以下に挙げる。

- ニューラルネットの入力層は、ベクトルではなく Convolution2D を用いて行列の

まま、入力値を扱う。

- ニューラルネットの中間層は、三次元テンソルとして扱う。

これらは、リバーシのルールによりベクトルとして扱うより、最善値収束が早いためこのような工夫を施した。

Chapter 5　Result

5.1　pre-Exp

5.1.1　pre-Exp1:Hyper Parametars

最適ハイパーパラメータの選定の結果を Figure 5.1、Figure 5.2、Figure 5.3 に示す。割引率を基準としたグラフ作成を行った。

Discussion

まず Figure 5.1 より、最速で収束する学習率が 0.0001 であることが分かる。これは 201episode 目で、直前 10episode での平均値が 195step を超えた。学習率が 0.001 のときは、episode が 20〜30 の間に、収束するような傾向があるもののすぐ発散してしまい、以後 1000episode 経つまで収束はしなかった。これは、学習率が大きすぎるため本来収束すべき最適値を超えて、局所解に陥ってしまったためだと考えられる。それは学習率を 0.01 に変化させた際に、全く収束する素振りを見せなかったことからも同様に結論付けることができる。学習率 0.0001 では、180episode 以後徐々に収束するような過程が見られた。DQN の差分という考えと微分の考えから、本来の学習率は 0 に近ければ近いほど、微分の定義に近づくため収束しやすいと考えられる。対して、学習率が小さすぎると、1episode ごとの変化量も小さいため、収束がかなり遅くなってしまうことも分かる。

次に Figure 5.2 より、最速で収束する学習率が 0.001 であることが分かる。これは 123episode 目で、直前 10episode での平均値が 195step を超えた。同様にして、学習率が 0.01 では 50episode まで収束する素振りを見せるものの、以降局所解に陥ってしまった。ただ他の学習率に対しても言えるが、一度収束から外れた後、再度収束値へ向かう振る舞いを見せたため、学習率が小さければ小さいほど、収束性が高まることが分かる。

最後に Figure 5.3 より、最速で収束する学習率が 0.001 であることが分かる。これは 46episode 目で、直前 10episode での平均値が 195step を超えた。これは、すべての学習率において収束した事がわかる。割引率が低い場合においては、未来の影響が小さくなるため局所解に陥る可能性が低い。また、学習率が 0.01 のときも何度か局所解に陥る場面が存在したが、真の最適解へと収束するように軌道修正ができていたため、割引率は 0.95 など 1.0 により近くなくても良いことがわかった。

これらの結果より最適値を確定させ、本実験では学習率を 0.001 、割引率を 0.95 と

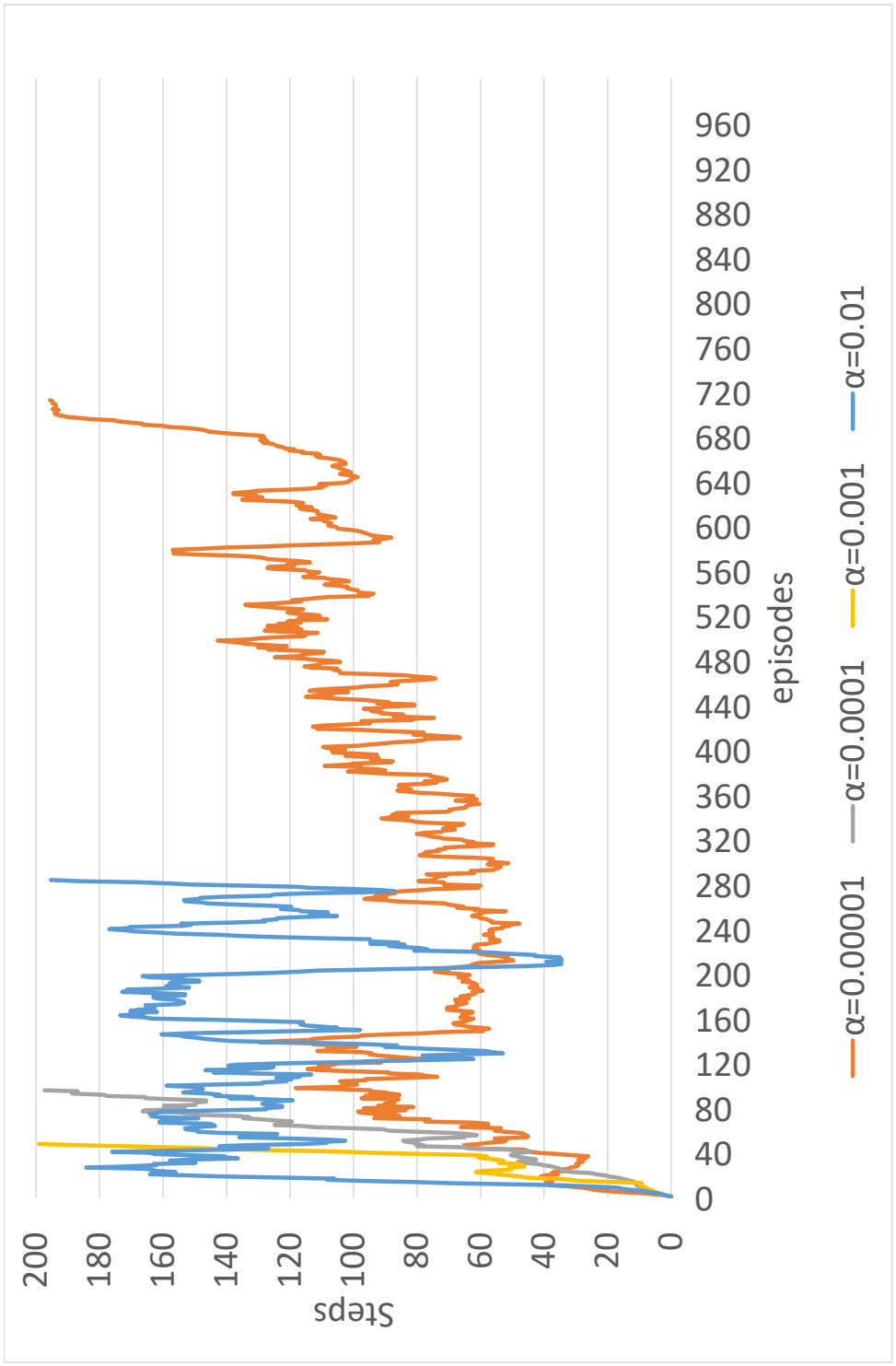

Figure 5.1 $\gamma = 0.95$

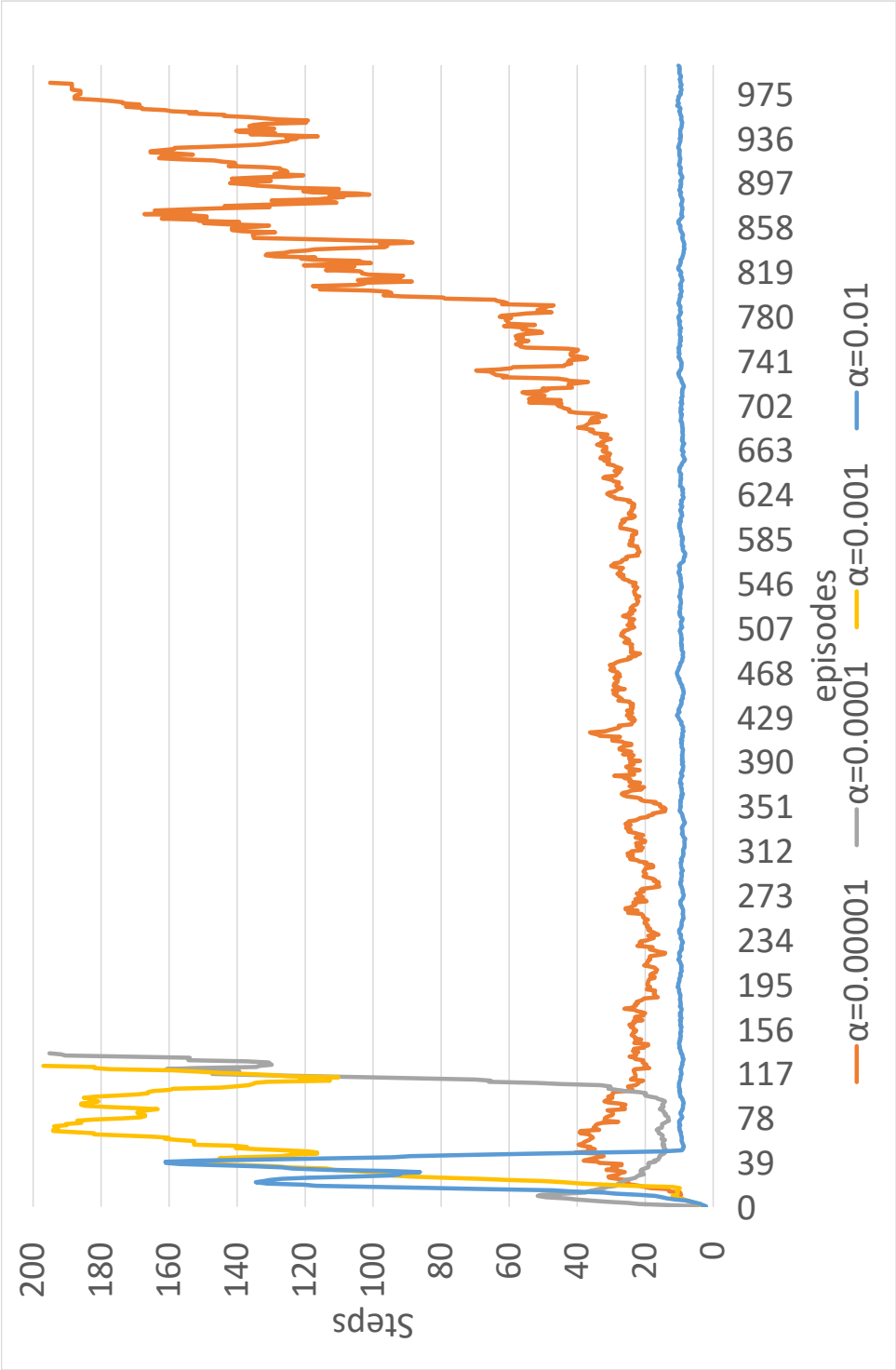

Figure 5.2 $\gamma = 0.97$

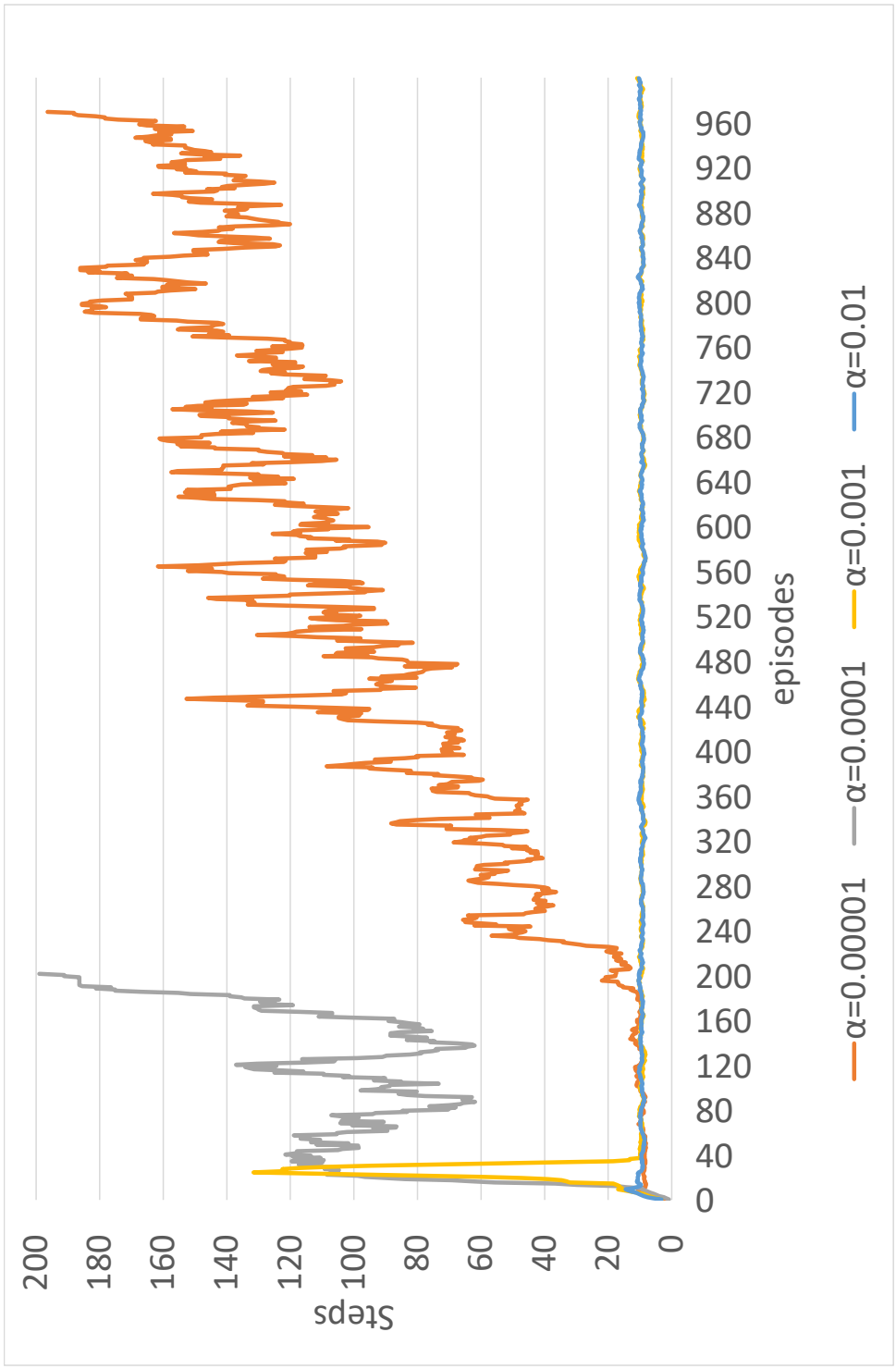

Figure 5.3 $\gamma = 0.99$

した。

5.1.2 pre-Exp2:Networks

最適ネットワークサイズの選定の結果を Figure 5.4 に示す。

Discussion

Figure 5.4 より、3 層では 100episode あたりまでの収束経過が遅い。4 層では、46episode で収束したためこの単純なモデルにおいては十分な層数であると言える。5 層では初期の収束がとても早いが、150step 程度での停滞が長く続いたため、直前 10episode での平均値 195step 以上になるには 4 層の 10 倍弱時間がかかってしまった。しかし、通常であれば局所解に陥り 10step 前後で停滞し以後収束することがないため、5 層での学習は 150step 程度で停滞していることから、綿密な重み学習を繰り返していると考えられる。つまり、8x8 のリバーシやそれ以上の複雑な問題における回帰には、有効な手段ではないかと考えられる。

この結果より、本実験では 4 層ではなく、5 層を用いることとした。

5.1.3 pre-Exp3:Neurons

最適ニューロン数の選定の結果を Figure 5.5 に示す。

Discussion

Figure 5.5 より、ニューロンの数は 8-16 のときにかなり収束がふらつき非常に遅くなったが、ほかは比較的早く、16-32 や 32-16、32-32 は 30episode を超える前に収束していた。学習が偶然上手く行ったのではないかと考え、何度か同じニューロン割当てで実験を行ったが、あまり大きな変化はなく早く収束したため、確認のため 64-64 などのニューロン数に変化させたが、32 以上の数にすると収束が遅く不安定になってしまったため、入力に対して 10 倍程度のニューロン数を中間層に用いることが良いのではないかと考えられる。

この結果より、本実験では入力が 64 になるので、中間層を 640-640-640 として行った。

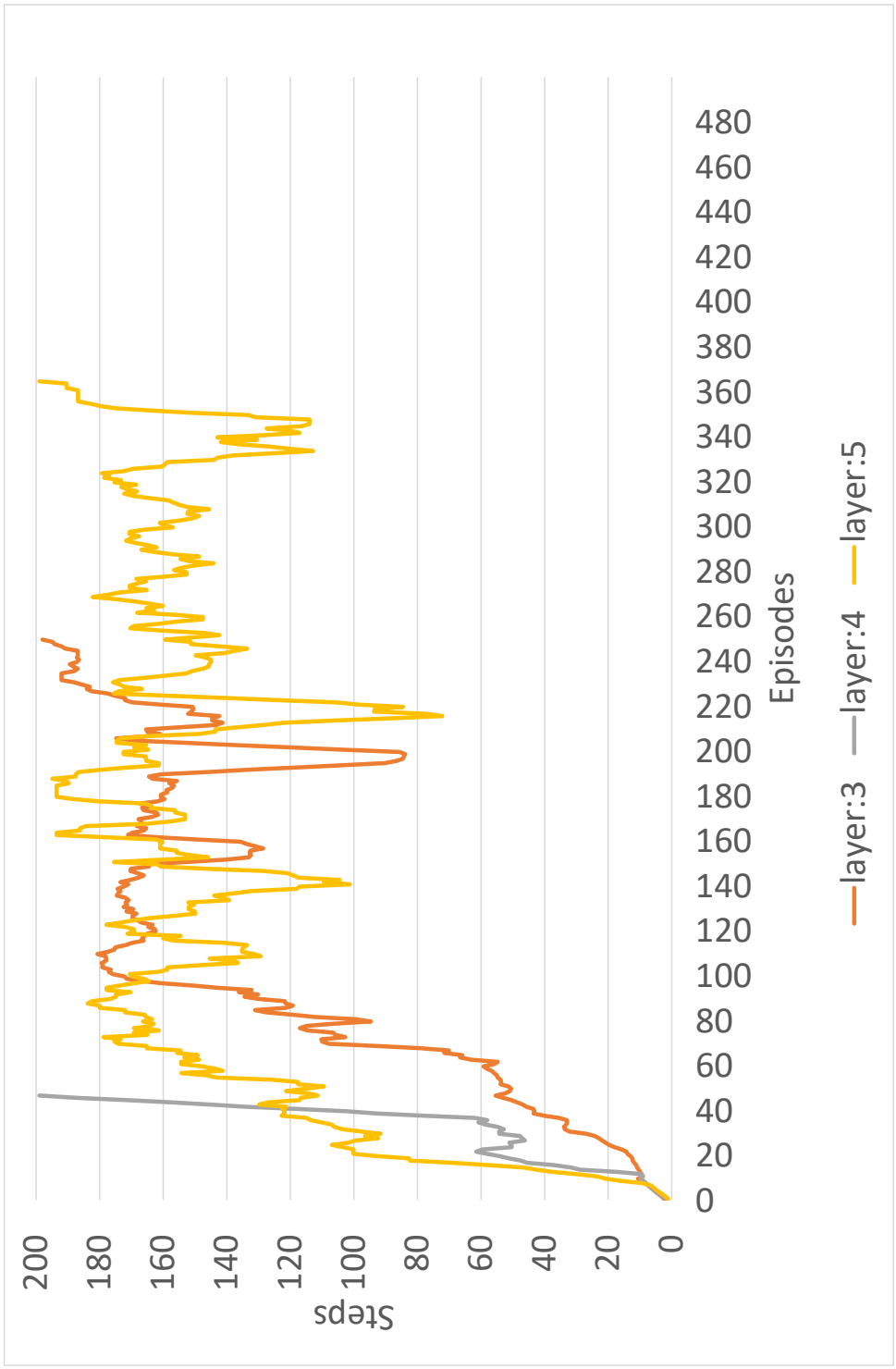

Figure 5.4 The size of networks

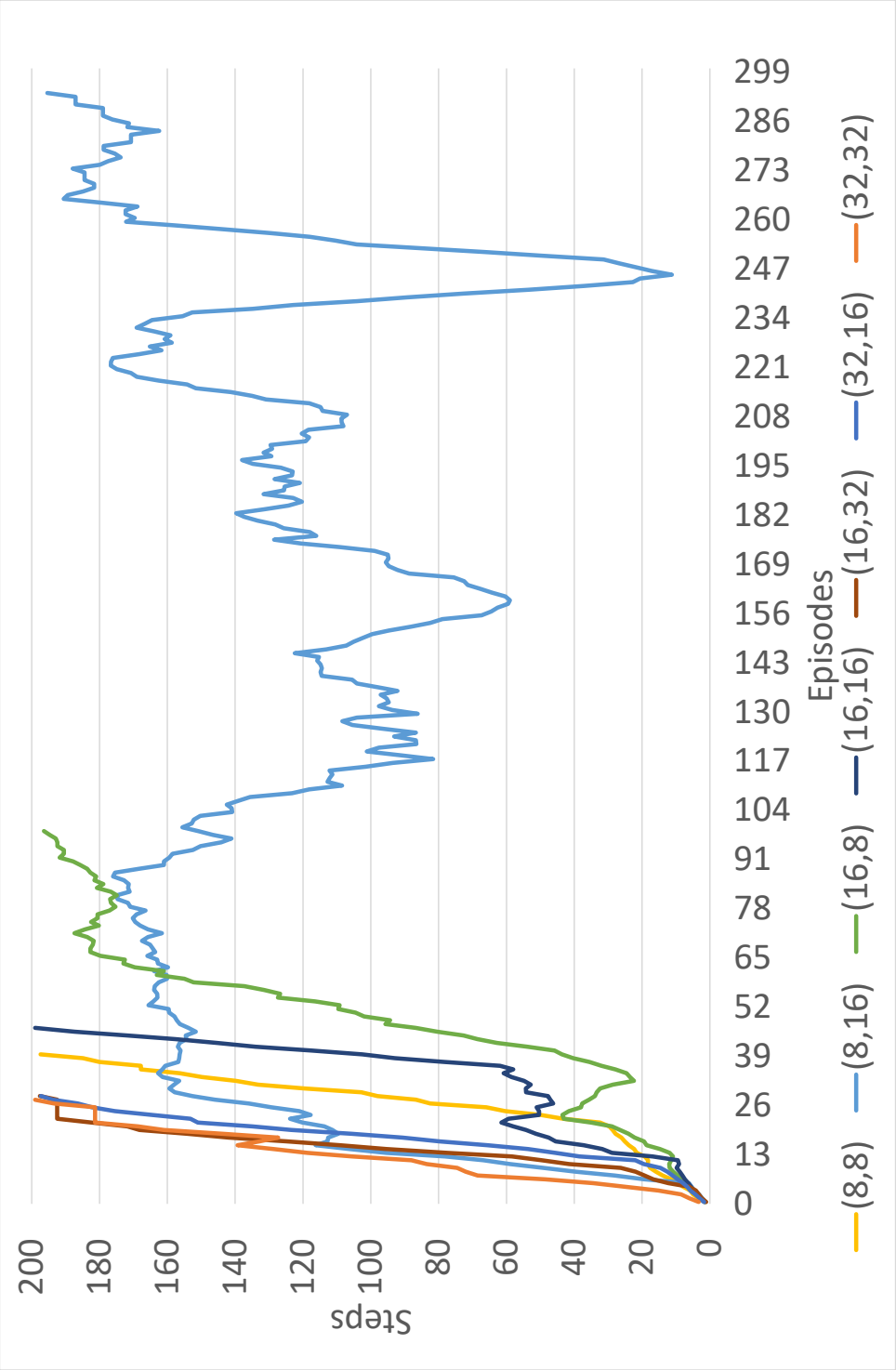

Figure 5.5 The number of neurons

Table 5.1　DQN AI's win rate

episode	MTS:100	Random
0	0	
1	0.3	0.8
2	0.5	0.8
3	0.2	1.0
4	0.7	0.9
5	0.3	0.9
6	0.3	0.9
7	0.4	0.9
8	0.4	1.0
9	0.4	0.9

5.2　Experiment:Reversi AI

8x8 リバーシへ DQN アルゴリズムに適用した結果を Table 5.1 に示す。ここで、第 0 世代とはランダム AI を指している。

Discussion

Table 5.1 より MTS:100 AI を DQN AI と対戦させたとき、第 4 世代のとき勝率 0.70 と最高の勝率を得ることができた。しかし、それ以外の世代ではほぼ互角となった。またこの世代においては、リバーシにおける序盤の配置である 4 定石に置いている傾向が多々見られたため、ルールや戦略を事前に学習させずとも自然に学習するといえる。ランダム AI と対戦させたときは、第 3 世代以降すべての世代の AI において、勝率 0.9 を超えている。

これらの結果から、事前実験での学習失敗例に似た結果が第 5 世代位以降で出てしまったため、学習が失敗してしまったのではないかと考えられる。原因は NFQ によるもので、ハイパーパラメータの設定やニューラルネットワークの設定が不適であったとは考えにくい。NFQ は事前に用意した譜面に依存する傾向があるため、その譜面の学習を繰り返したことで学習が途中で失敗してしまったのではないかと考えられる。第 4 世代ま

での学習は、勝率の変化からかなり良く学習できていたので、事前実験で考察した最適値の適用は、間違いではなかったともいえる。

Chapter 6　Conclusion

本研究では、通常の 8x8 リバーシに DQN と呼ばれる性能の良い畳み込みニューラルネットワークと Q 学習を用いた回帰アルゴリズムを実装することを試みた。また、8x8 リバーシに DQN を実装する前に、あまり明らかにならないハイパーパラメータの役割と最適値を調べるため、事前実験として、学習率・割引率・ニューラルネットワークの隠れ層数・ニューロンの個数について個別で実験し、考察を行った。結果として、学習率の変化は小さくても、大きく結果の収束時間や真最適解への収束率に影響を与えてしまうため、慎重に選択すべきハイパーパラメータであると考察できた。また、8x8 リバーシへの DQN の実装における DQN の利用を演算量の都合上とはいえ、NFQ で置き換えなければならない部分が発生してしまったため、初期譜面の 26 万譜面数をもってしても、モンテカルロ木探索アルゴリズムとほぼ互角であった結果が出てしまったことから、演算専用の Tensor Processor Unit を用いて DQN のみを用いた学習をすべきだという結論に至った。Google には Google Colab と呼ばれる、無料の AI 開発環境があるため、このようなサービスを利用すれば、より強力な AI を作成できたのではないかといえる。今回の実験で用いたニューラルネットワークは、いずれも 5 層程度の比較的浅めな深層ニューラルネットワークであったが、事前実験の結果から、より深い層を持つニューラルネットワークは、振る舞いの複雑さにより演算量の多い環境において効果的であると予想される。

そして DQN は新しいといっても 2013 年に提案された手法であり、インターネット上・コンピュータ科学研究・人工知能研究分野では常に新しく強力なアルゴリズムが提案され続けている。2018 年では上位互換となる Rainbow や Ape-X などの応用アルゴリズムが提案されているため、これら最新のアルゴリズムを同様にリバーシやそれよりも演算量の多い将棋・囲碁などへ実装したら、どれだけの違いが生まれるのかという実験・考察を行いたいと考えている。

今回の実験では、ニューラルネットワークの構築に TensorFlow の Keras ライブラリを用いたが、簡単に層追加やニューロン個数変更ができたため、実験をスムーズに行うことができた。また簡単に利用できる点から、バグが発生しにくいという利点がある。AI 開発にはこういった機械学習向けライブラリが多く存在するが、このようなライブラリと新たな手法を組み合わせると、より簡単に強力な AI を作成することが可能になる。そ

のため今後のAI開発には、様々な分野で気軽に活用できるようになることを願いたい。

Acknowledgements

本論文は筆者が岐阜工業高等専門学校電気情報工学科に在籍中の研究成果をまとめたものである。同学科教授出口先生には指導教官として本研究の実施の機会を与えていただき、その遂行にあたって終始、ご指導をいただいた。ここに深謝の意を表する。同学科教授安田先生には副査としてご助言いただくとともに本論文の細部にわたりご指導いただいた。ここに深謝の意を表する。本学科出口研究室の各位には研究遂行にあたり日頃より有益なご討論ご助言をいただいた。ここに感謝の意を表する。

Bibliography

1) 小林 一郎 (2008) 『人工知能の基礎』 サイエンス社.

2) 舩橋 聡太 (2018) “ディープラーニングを用いたパーフェクト・リバーシの局面評価の研究” 岐阜工業高等専門学校電気情報工学科卒業研究報告.

3) Richard S.Sutton , Andrew G.Barto (2000) 『強化学習』 三上 貞芳, 皆川 雅章共訳 森北出版株式会社.

4) Riedmiller, Martin. (2005) “Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method.” European Conference on Machine Learning. Springer, Berlin, Heidelberg.

5) Mnih, Volodymyr, et al. (2013) “Playing atari with deep reinforcement learning.” arXiv preprint arXiv:1312.5602.

6) 小川 雄太郎 (2018) 『つくりながら学ぶ！深層強化学習 PyTorch による実践プログラミング』 マイナビ出版.

7) Andreas C. Muller, Sarah Guido (2017) 『Python ではじめる機械学習 ——scikit-learn で学ぶ特徴量エンジニアリングと機械学習の基礎』 中田 秀基訳 第 5 刷 O’Reilly Japan.

8) 有山 圭二 (2016) 『Tensorflow はじめました　実践！最新 Google マシンラーニング』 インプレス R&D.