卒 業 研 究 報 告 題 目

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

A Study of Evaluation Function for Perfect Reversi by Deep Learning

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

2013E33 船橋 聡太

平 成 ３ ０ 年（ ２ ０ １ ８ 年 ）　　２ 月 １ ６ 日 提 出

Abstract

A field of AI, stands for Artificial Intelligence, has been rapidly developed since it was appeared in Dartmouth Conference in 1956. AI is frequently used in every field such as Natural Language Processing, Image Recognition and Expert System, and so on. Specifically, in AI, Neural Network Theory is spotlighted in recent years since Hierarchical Neural Network achieved exceptional performance in regression and classification problems. In this research, using 10×10 squares reversi, so called "Perfect Reversi", its AI is trained by deep learning using existing training data sets. With TensorFlow library that is appeared recently for the machine learning, Convolutional Neural Network is constructed so as to train the perfect reversi AI. After that, through the matches of the trained AI and random AI, its performance is evaluated. Since the perfect reversi would be the game which have a lot of uncertainties, there are lots of difficulties in determination of evaluation value and optimization method.

目 次

第1章　序論

人工知能（AI）という分野は、1956年に行われたダートマス会議の際に誕生して以来、今日まで急速な発展を続けている。その応用領域は多岐に渡り、自然言語処理や音声・画像認識、エキスパートシステムなど現実世界のあらゆる分野で利用されている。ニューラルネットワークは、人工知能分野の中でも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、層を何重にも重ねて構築した階層型ニューラルネットワークを用いた学習、ディープラーニング（深層学習）が回帰問題や分類問題に於いて非常に優れた性能を示したためである。

本研究では、パーフェクト・リバーシと呼ばれる 10×10 マスの大盤リバーシの局面評価にこのディープラーニング手法を用いて、既存の訓練データセットによる学習を試みる。学習に用いる畳み込みニューラルネットワークの構築には、近年登場した新しい機械学習向けライブラリであるTensorFlowを用い、制作したAIとランダムで駒を打つAIとの性能を比較検討する。

第2章　ニューラルネットワーク

2.1　ニューロン

ニューロンとは、Figure2.1 に示すような生体の神経系を構成する神経細胞で、その機能は情報処理系及び情報伝達に特化しているという特徴を持つ。ニューロンの主たる機能は、入力刺激を受け取った際に活動電位を発生させ、他の細胞に信号を伝達することである。成人の脳に於いては約 1000 億個のニューロンが存在し、3 次元的に密結合した回路網を形成している。

ニューロンは、その本体の部分となる細胞体と、複雑に枝分かれした樹状突起、本体から 1 本だけ出ていて末端で多数に枝分かれする軸索と呼ばれる 3 つに分かれている。情報は軸索と樹状突起を通じて、ニューロンからニューロンへ伝達される。軸索は、活性化されたニューロンからそのニューロンにつながっている他のニューロンへ電位または活動電位を伝え、この活動電位を樹状突起の受容体が受け取ることでシナプスの隙間で化学反応が起こり、特定のニューロンへの活動電位入力が励起または抑制される。このような情報の伝達が並列処理で行われるため、全体として高い情報処理能力を誇っている [1)]。

2.2　ニューロンモデル

ニューロンには多数の種類の存在が確認されており、単純な発火作用を持つものから非常に複雑な発振作用を持つものまで存在している。しかし、一般的なニューラルネットワークで用いられているニューロンモデルは、Figure2.2 のように単純化された複入力・1 出力のモデルが通常である。ニューロンの出力は、式 (2.1) で表される。w_i は入力値 x_i に対する結合荷重であり、b はバイアス項である。

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

2.3　活性化関数

活性化関数は別名伝達関数とも呼ばれ、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若しくは恒等関数のことを指す。活性化関数として用いられる関

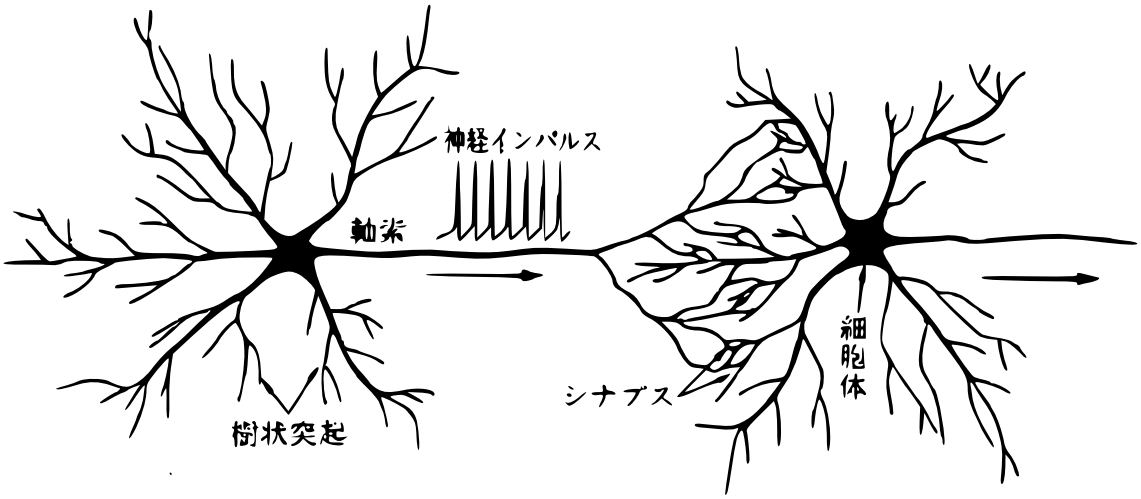

Figure 2.1　Neuron[2)]

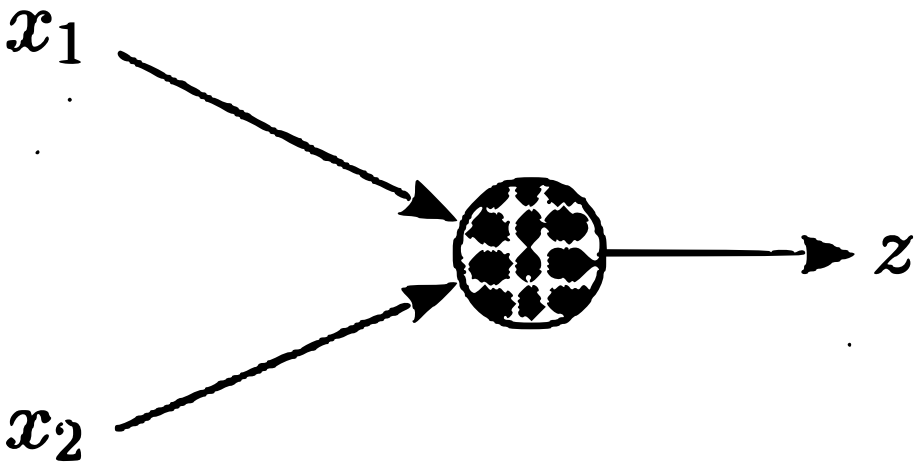

Figure 2.2　Neuron model[3)]

数は様々で、代表的なものを次に示す。

- ステップ関数（Figure2.3）
- シグモイド関数（Figure2.4）
- ReLU 関数（Figure2.5）

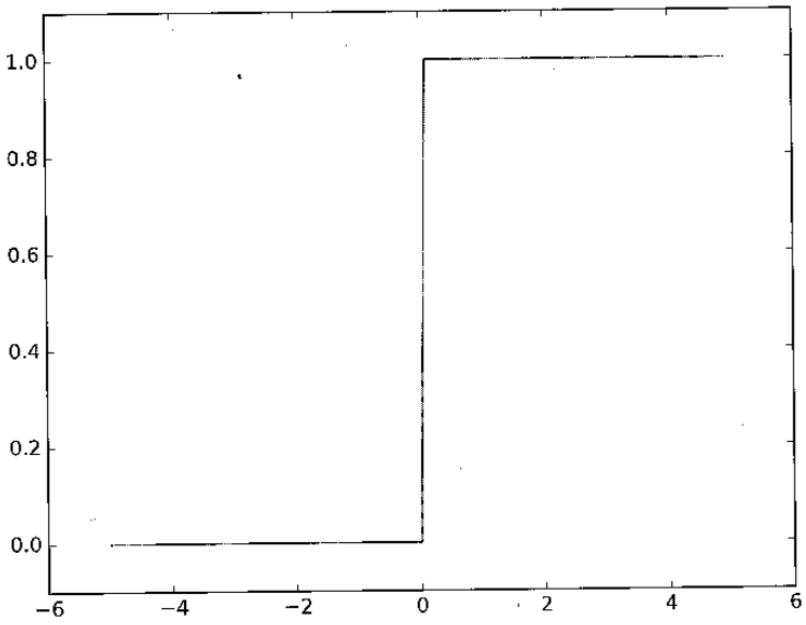

Figure 2.3　Step function[4)]

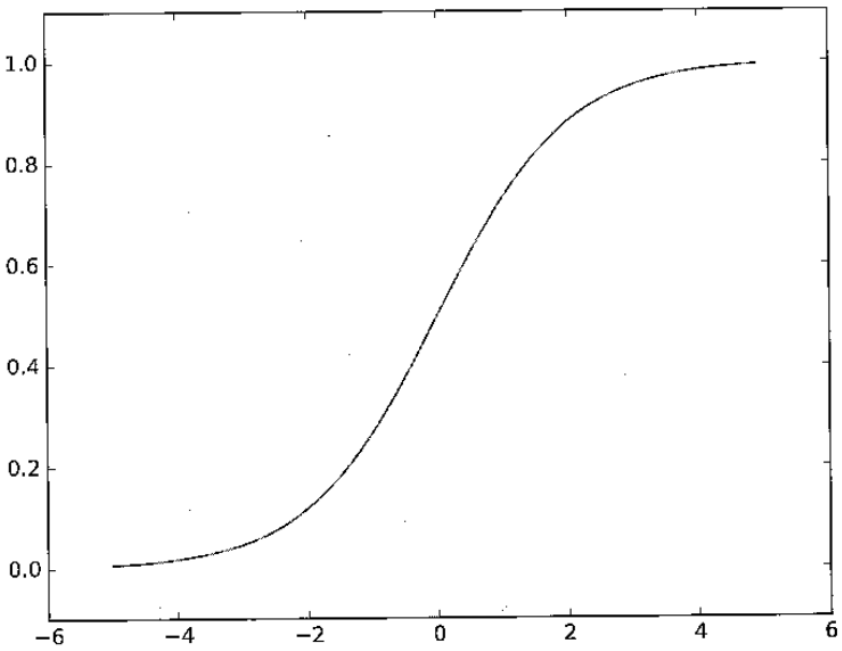

Figure 2.4　Sigmoid function[4)]

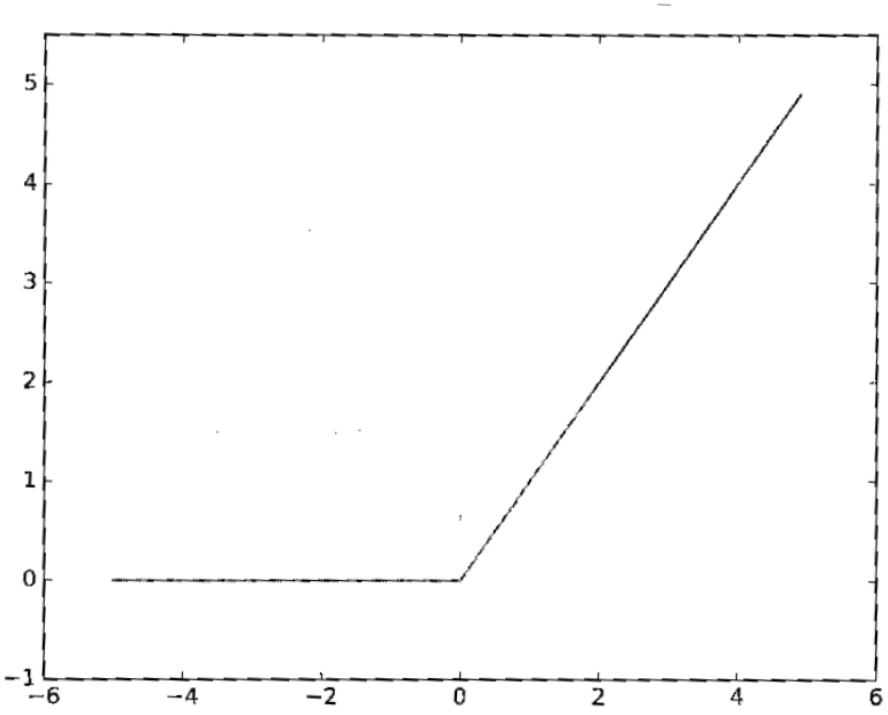

Figure 2.5　ReLU function[4)]

第3章　パーセプトロン

3.1　パーセプトロン

パーセプトロンとは、1957年にアメリカの心理学者･計算機科学者であるFrank Rosenblattが考案したアルゴリズムである。パーセプトロンは複数の二値信号を入力として受け取り単一の二値信号を出力するアルゴリズムで、パーセプトロンを用いてANDゲートやORゲートなどの論理回路を実現することが可能である。このような特徴を持つパーセプトロンは、ニューラルネットワーク及びディープラーニングの起源となるアルゴリズムである。

3.2　単層パーセプトロン

単層パーセプトロンは、別名「人工ニューロン」や「単純パーセプトロン」とも呼ばれるアルゴリズムである。単層パーセプトロンを用いれば、ANDゲートやNANDゲート・ORゲートなどの論理回路を表せるが、XORゲートのような論理回路は表せないという欠点がある。これは、単層パーセプトロンが非線形分離問題を解決できないことに起因する。対象とする問題が線形分離問題であれば、パーセプトロンの収束定理により解決できることが証明されている。単層パーセプトロンの例をFigure3.1に示す。単層パーセプトロンにおける結合荷重やバイアスの値は、様々な学習法を用いて学習させることが可能である。

3.3　多層パーセプトロン

パーセプトロンはそれらの層を重ね合わせることで、新たな1つのパーセプトロンを構築することが可能である。このように構築された2層以上のパーセプトロンを「多層パーセプトロン」と呼ぶ。多層パーセプトロンは、単層パーセプトロンに存在していた非線形分離問題を解決できないという欠点を克服しており、適切にパーセプトロンを構築すれば理論上は複雑な関数や非線形関数を表現することが可能である。そのような特徴を持つ多層パーセプトロンでは、単層パーセプトロンでは表現不可能であったXORゲートなどの論理回路を表現することが可能であるが、多層パーセプトロンにおける結合荷重やバイアスなどの値は、単層パーセプトロン同様に様々な学習法を用いて学習させることが可能である。多層パーセプトロンの例をFigure3.2に示す。

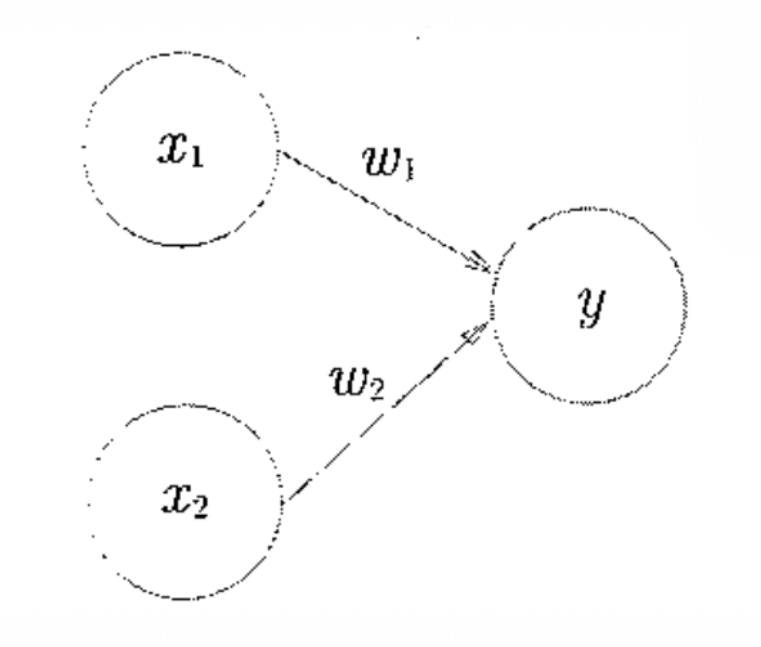

Figure 3.1 Single layer perceptron[4)]

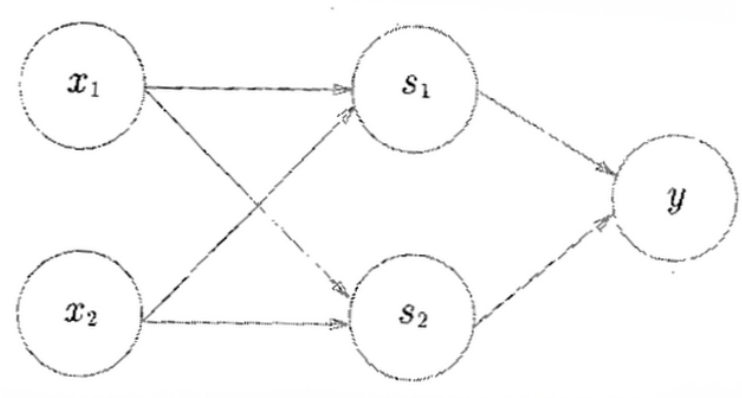

Figure 3.2 Multilayer perceptron[4)]

第4章　ディープラーニング

ディープラーニング（深層学習）とは、多層化されたニューラルネットワークを用いた機械学習手法を指す。ディープラーニングの登場以前、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技術的問題が存在し、実用化が困難であった。しかし、カナダのコンピュータ科学・認知心理学者 Geoffrey Everest Hinton らによる多層ニューラルネットワークの研究や、計算機の性能向上などの理由により、2010年代に深い層のニューラルネットワークを用いた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
- 敵対的生成ネットワーク
- ボルツマンマシン
- 制約ボルツマンマシン

4.1　畳み込みニューラルネットワーク

畳み込みニューラルネットワークとは、順伝播型人工ディープニューラルネットワークの一種である。主に画像認識や動画認識・音声認識などの分野で広く用いられており、特に画像認識のコンペティションでは、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。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演算に相当する。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。

第5章　学習方法

5.1　最急降下法

最急降下法とは、連続最適化問題における勾配法に分類されるアルゴリズムで、関数の一階微分値を参照してその関数の最小値を探索する手法である。尚、名前の由来は最も傾きの急な方向に降下することを意味しており、収束の速さについて言及している訳ではない。最急降下法には、以下のような特徴がある。

- 簡便で計算速度が速い
- 他の手法に比べ、局所的な最小値に収束する可能性が高い

5.2　確率的勾配降下法

確率的勾配降下法とは、最急降下法と同様に連続最適化問題における勾配法に分類されるアルゴリズムであるが、最小値の探索を乱択的に行う点が最急降下法と異なる。確率的勾配降下法による学習は、ミニバッチ学習による最急降下法によって実現できる。確率的勾配降下法には、以下のような特徴がある。

- 局所的最小値に収束する可能性が最急降下法よりも低い
- 入力値に平均や分散が極端に異なるデータが存在する場合、探索が成功しにくくなる

5.3　誤差逆伝搬学習法

誤差逆伝搬学習法とは、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年にアメリカの認知心理学者 David E. Rumelhart らが考案した機械学習手法である。誤差逆伝搬学習法には、以下のような特徴がある。[5)]

- 活性化関数は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と、勾配消失問題が生じやすくなる

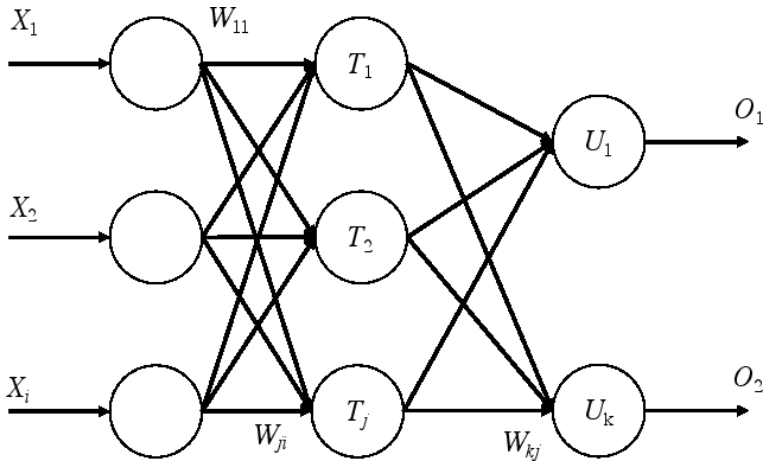

Figure 5.1　Backpropagation[5)]

誤差逆伝搬学習法による学習は、結合荷重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。まず、Figure5.1 中で用いられている変数について説明を加える。W_{ij} とは、入力層と中間層間の結合荷重、W_{kj} は中間層と出力層間の結合荷重である。T_j は中間層のニューロンの出力値で、式 (5.1) で表される。

$$T_j = \sum_{i=1}^{n} W_{ji} X_i \tag{5.1}$$

U_k は出力層のニューロンに入力される値で、中間層と同じようにして求めることができる。出力層の i 番目のニューロンの出力値 O_i に対して、t_i はその教師信号である。次に学習方法について説明する。学習の際に必要な誤差 E について定義する。E の定義式は式 (5.2) に示す。

$$E = \frac{1}{2} \sum_{i=1}^{n} (t_i - O_i)^2 \tag{5.2}$$

バックプロパゲーションの学習においては、この誤差が小さくなるように結合荷重を修正していく。まずは、中間層-出力層の結合荷重の学習を見ていく。結合荷重 W_{kj} に対する誤差の変化を見ればいいので、E を W_{kj} で偏微分すれば良い。この計算式を式 (5.3)

に示す。

$$\begin{aligned} \triangle W_{kj} &= -\eta \frac{\partial E}{\partial W_{kj}} \\ &= -\eta \frac{\partial E}{\partial O_k} \frac{\partial O_k}{\partial U_k} \frac{\partial U_k}{\partial W_{kj}} \end{aligned} \tag{5.3}$$

η は学習係数であり、学習 1 回当たりの結合荷重の変化量を示す。大きすぎると学習が大雑把になり、小さすぎると学習に膨大な回数がかかるようになるため、学習係数の値は適切に設定する必要がある。学習係数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。まずは、それぞれの微分の計算をする。誤差を出力で微分する式を式 (5.4) に示す。

$$\begin{aligned} \frac{\partial E}{\partial O_k} &= \frac{\partial}{\partial O_k} \frac{1}{2} (t_k - O_k)^2 \\ &= -(t_k - O_k) \end{aligned} \tag{5.4}$$

次に、出力を偏微分する式を式 (5.5) に示す。

$$\begin{aligned} \frac{\partial O_k}{\partial U_k} &= \frac{\partial}{\partial U_k} \frac{1}{1 + e^{-U_k}} \\ &= \frac{e^{-U_k}}{(1 + e^{-U_k})^2} \\ &= \frac{1}{1 + e^{-U_k}} \left(1 - \frac{1}{1 + e^{-U_k}} \right) \\ &= O_k (1 - O_k) \end{aligned} \tag{5.5}$$

最後に、出力値を結合荷重で偏微分する式を式 (5.6) に示す。

$$\begin{aligned}\frac{\partial U_k}{\partial W_{kj}} &= \frac{\partial}{\partial W_{kj}}\left(H_1 W_{k1} + H_2 W_{k2} + ... + H_j W_{kj}\right) \\ &= H_j\end{aligned} \tag{5.6}$$

以上の式より、修正量 $\triangle W_{kj}$ は式 (5.7) の形で表される。

$$\begin{aligned}\triangle W_{kj} &= -\eta \frac{\partial E}{\partial W_k j} \\ &= -\eta \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial U_k}\frac{\partial U_k}{\partial W_{kj}} \\ &= -\eta\{-(t_k - O_k)O_k(1 - O_k)H_j\} \\ &= \eta \delta_k H_j\end{aligned} \tag{5.7}$$

ここで、δ_k は、式 (5.8) と定義される。

$$\delta_k = (t_k - O_k)O_k(1 - O_k) \tag{5.8}$$

出力層の結合荷重は、以上のようにして算出できる。次に、入力層と中間層間の結合荷重の計算について説明する。結合荷重の修正量は、中間層と出力層間と同様に、式 (5.9) で算出できる。

$$\triangle W_{ji} = -\eta \frac{\partial E}{\partial W_{ji}} \tag{5.9}$$

出力層の結合荷重の更新には k 番目の出力 O_k のみが寄与していたが、中間層の結合荷重の更新には n 個の出力全てが寄与する。この偏微分は、連鎖律（チェイン・ルール）を用いて計算を行う。中間層の結合荷重の修正量 $\triangle W_{ji}$ は式 (5.10) で算出できる。

$$\begin{aligned}\triangle W_{ji} &= -\eta \frac{\partial E}{\partial W_{ji}} \\ &= -\eta \left(\sum_{i=1}^{n} \frac{\partial E_i}{\partial O_i}\right)\frac{\partial H_j}{\partial T_j}\frac{\partial T_j}{\partial W_{ji}}\end{aligned} \tag{5.10}$$

出力値 U_i を中間層の出力 H_j で偏微分する式を式 (5.11) に示す。

$$\begin{aligned}\frac{\partial U_i}{\partial H_j} &= \frac{\partial}{\partial H_j}(H_1W_{k1} + H_2W_{k2} + ... + H_nW_{kn}) \\ &= W_{kj}\end{aligned} \tag{5.11}$$

中間層の出力 H_j を出力値 T_j で偏微分する式を式 (5.12) に示す。

$$\begin{aligned}\frac{\partial H_j}{\partial T_j} &= \frac{\partial}{\partial T_j}\left(\frac{1}{1+e^{-T_j}}\right) \\ &= -\frac{e^{-T_j}}{1+e^{-T_j}} \\ &= \frac{1}{1+e^{-T_j}}\left(1 - \frac{1}{1+e^{-T_j}}\right) \\ &= H_j(1-H_j)\end{aligned} \tag{5.12}$$

出力値 T_j を結合荷重 W_{ji} で偏微分する式を式 (5.13) に示す。

$$\begin{aligned}\frac{\partial T_j}{\partial W_j i} &= \frac{\partial}{\partial W_{ji}}(X_1W_{j1} + X_2W_{j2} + ... + X_nW_{jn}) \\ &= X_i\end{aligned} \tag{5.13}$$

これらの式より、結合荷重の修正量は式 (5.14) のようになる。

$$\begin{aligned}
\triangle W_{ji} &= -\eta \frac{\partial E}{\partial W_{ji}} \\
&= -\eta \frac{\partial H_j}{\partial T_j} \frac{\partial T_j}{\partial O_i} \frac{\partial O_i}{\partial U_i} \frac{\partial U_i}{\partial H_j} \\
&= \eta H_j (1 - H_j) X_i \sum_{k=1}^{n} W_{kj} (t_k - O_k) O_k (1 - O_k) \\
&= \eta H_j (1 - H_j) X_i \sum_{k=1}^{n} W_{kj} \delta_k \\
&= \eta \delta_j X_i
\end{aligned} \tag{5.14}$$

ここで、δ_j は、式 (5.15) と定義される。

$$\delta_j = H_j (1 - H_j) \sum_{k=1}^{n} W_{kj} \delta_k \tag{5.15}$$

中間層の結合荷重は、以上のようにして算出する。

5.4 AdaGrad 法

AdaGrad 法は 2011 年に John Duchi らによって考案された機械学習手法であり、以下のような特徴がある。[3)]

- 学習係数を自動で調整する
- 初期学習係数 η_0 を定める必要がある

なお、学習に AdaGrad 法を用いる際は、結合荷重の値が無限大に発散するのを防ぐために、ϵ パラメータの値は小さな正の定数を設定しなければならない。AdaGrad 法によるパラメータ修正量の計算式を式 (5.16)、式 (5.17)、式 (5.18)、式 (5.19) に示す。

$$h_0 = \epsilon \tag{5.16}$$

$$h_t = h_{t-1} + \nabla Q_i(w) \circ \nabla Q_i(w) \tag{5.17}$$

$$\eta_t = \frac{\eta_0}{\sqrt{h_t}} \tag{5.18}$$

$$w^{t+1} = w^t - \eta \nabla Q_i(w^t) \tag{5.19}$$

学習を重ねる毎に、学習係数 η_t の値は小さくなっていくことが分かる。AdaGrad 法では、η_t の値が 0 に漸近し目的解へ収束する。尚、Chainer ライブラリにおいては、デフォルトのパラメータ値は式 (5.20)、式 (5.21) のように設定されている。

$$\epsilon = 10^{-8} \tag{5.20}$$

$$\eta_0 = 0.001 \tag{5.21}$$

5.5 Adam 法

Adam 法は 2015 年に Diederik P. Kingma らによって考案された機械学習手法であり、その理論はやや複雑である。Momentum や AdaGrad を融合した手法で、効率的にパラメータ空間を探索することが期待できる。Adam 法には、以下のような特徴がある。[3)]

- 誤差逆伝搬学習法よりも精度及び収束速度が速い
- バイアス補正が行われる

Adam 法による学習を行う時、ハイパーパラメータは式 (5.22)、(5.23)、(5.24)、(5.25) に設定する事が論文内で推奨されている。

$$\alpha = 0.001 \tag{5.22}$$

$$\beta_1 = 0.9 \tag{5.23}$$

$$\beta_2 = 0.999 \tag{5.24}$$

$$\epsilon = 10^{-8} \tag{5.25}$$

Adam 法によるパラメータ修正量の計算式を式 (5.26)、式 (5.27)、式 (5.28)、式 (5.29)、式 (5.30) に示す。

$$m_t + 1 = \beta_1 m_t + (1 - \beta_1)\nabla Q_i(w) \tag{5.26}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla Q_i(w) \circ \nabla Q_i(w) \tag{5.27}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{5.28}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{5.29}$$

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{5.30}$$

パラメータの初期値を式 (5.31)、(5.32) 式に示す。

$$m_0 = 0 \tag{5.31}$$

$$v_0 = 0 \tag{5.32}$$

Adam 法による学習では、誤差逆伝搬学習法などの学習法とは異なり、更新回数の少ないパラメータが優先的に更新される。これにより、最急勾配を選び優先的に下げていくのではなく、それ以外の方向にも探索を進めることで、理論上は局所的な最小値に陥る可能性が低くなる。

第6章　TensorFlow

6.1　TensorFlow

TensorFlowとは、Googleが2015年11月に公開したオープンソースの機械学習向けソフトウェアライブラリである。ディープラーニング対応のライブラリで、Googleの各種サービスなどに於いても幅広く活用されている。現段階ではC/C++, Python, Java, Goの4言語(5言語)に対応しており、マルチプラットフォームで動作する。また、TensorFlowにはグラフを可視化するTensorBoardが搭載されており、問題箇所の発見が容易になるという点で優れている。TensorFlowの仕組みは特徴的で、写像を定義することで計算グラフを作成していく。計算グラフの各ノードはオペレーションと呼ばれ、計算グラフの構築とセッションの実行に分けてプログラムを実装する[6)]。

6.2　TensorFlowの構文

6.2.1　グラフの構築

グラフ構築に使用するオペレーションの中で、頻繁に使用するオペレーションの作成方法を以下に示す。

- 定数オペレーション

  tensorflow.constant()
- 変数オペレーション

  tensorflow.Variable()
- 変数初期化オペレーション

  tensorflow.global_variables_initializer()
- プレースホルダ―オペレーション

  tensorflow.placeholder()
- シグモイド関数

  tensorflow.nn.sigmoid()
- ReLU関数

  tensorflow.nn.relu()
- Adam法

tensorflow.train.AdamOptimizer()

6.2.2 グラフの実行

グラフの実行方法を以下に示す。op は実行対象となるオペレーションである。

```
sess = tensorflow.Session()
sess.run(op)
sess.close()
```

6.2.3 グラフの保存

グラフの保存には以下の構文を用いる。sess は保存するセッションを示す。dir には保存先ディレクトリを記述する。

```
saver = tf.train.Saver()
saver.save(sess, dir)
```

6.2.4 グラフの復元

グラフの復元には以下の構文を用いる。sess は保存するセッションを示す。dir には保存先ディレクトリを記述する。

```
saver = tensorflow.train.Saver()
saver.restore(sess, dir)
```

6.3 MNIST

MNIST とは、「Mixed National Institute of Standards and Technology database」の略で、28 ピクセル x28 ピクセルの手書き文字画像に 0 から 9 までの数字の正解ラベルが与えられているデータセットである。訓練データ画像は 60,000 枚、テストデータ画像は 10,000 枚用意されており、画像認識の性能評価にしばしば用いられている。MNIST の一例を Figure6.1 に示す。

Figure 6.1　MNIST data examples[6)]

6.4　MNIST を用いた畳み込みニューラルネットワーク

6.4.1　TensorFlow のインポート

MNIST の学習には Python 用 TensorFlow を用いる。TensorFlow のクラスを利用するためには、以下のように tensorflow モジュールをインポートする必要がある。尚、学習の際は簡単のため Python の as 構文を用いてモジュール名を tf とリネームしておく。[5)]

```
import tensorflow as tf
```

6.4.2　結合荷重とバイアスの初期化

結合荷重の初期化とバイアスの初期化を行うメソッドを作成する。ローカルメソッドとして作成するため、第一引数に self は不要である。ここで、結合荷重については標準偏差 0.1 の切断正規分布、バイアスについては 0.1 の定数で初期化する。結合荷重とバイアスは変数であるため、tf.Variable で値を返す必要がある。TensorFlow において、Variable で構築した変数オペレーションは最適化の対象となる。[5)]

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

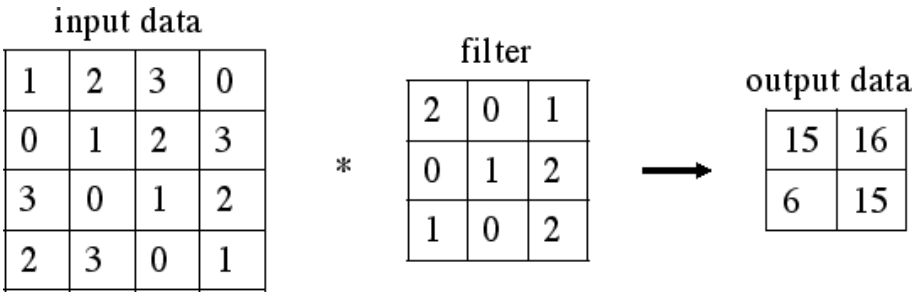

Figure 6.2　Convolution operation[5)]

6.4.3　畳み込み層の構築

畳み込み層を返すメソッドを作成する。畳み込み層で行う処理は畳み込み演算である。これは画像処理におけるフィルター演算に相当する。畳み込み演算の例を Figure 6.2 に示す。[5)]

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

Figure6.2 に示すように、畳み込み演算は入力データに対してフィルタを適用する。この例では、入力データは縦・横の次元を持つデータであるため、フィルタも同様に縦・横の次元を持つ。この例では、入力サイズが 4×4、フィルタサイズが 3×3、出力サイズが 2×2 となる。フィルタは、カーネルと呼ばれることもある。

Figure6.2 の畳み込み演算の例における実際の演算内容を Figure6.3 に示す。畳み込み演算は、入力データに対してフィルタのウィンドウを一定の間隔でスライドさせながら適用していく。ウィンドウとは、Figure6.3 における灰色の 3×3 の部分を指す。Figure6.3 に示すように、それぞれの場所でフィルタ要素と入力の対応する要素を乗算しその和を求める。そして、その結果を出力の対応する場所に格納していく。このプロセスをすべての場所で行うことで、畳み込み演算の出力を得ることができる。

第 1 引数は入力データを示し、[データ数、入力データの高さ、入力データの幅、入力チャンネル数] の 4 次元で表される。第 2 引数はフィルタである。 第 3 引数はストライドであり、フィルタを適用する位置の間隔を示している。ストライドは [1, 縦ストライド,

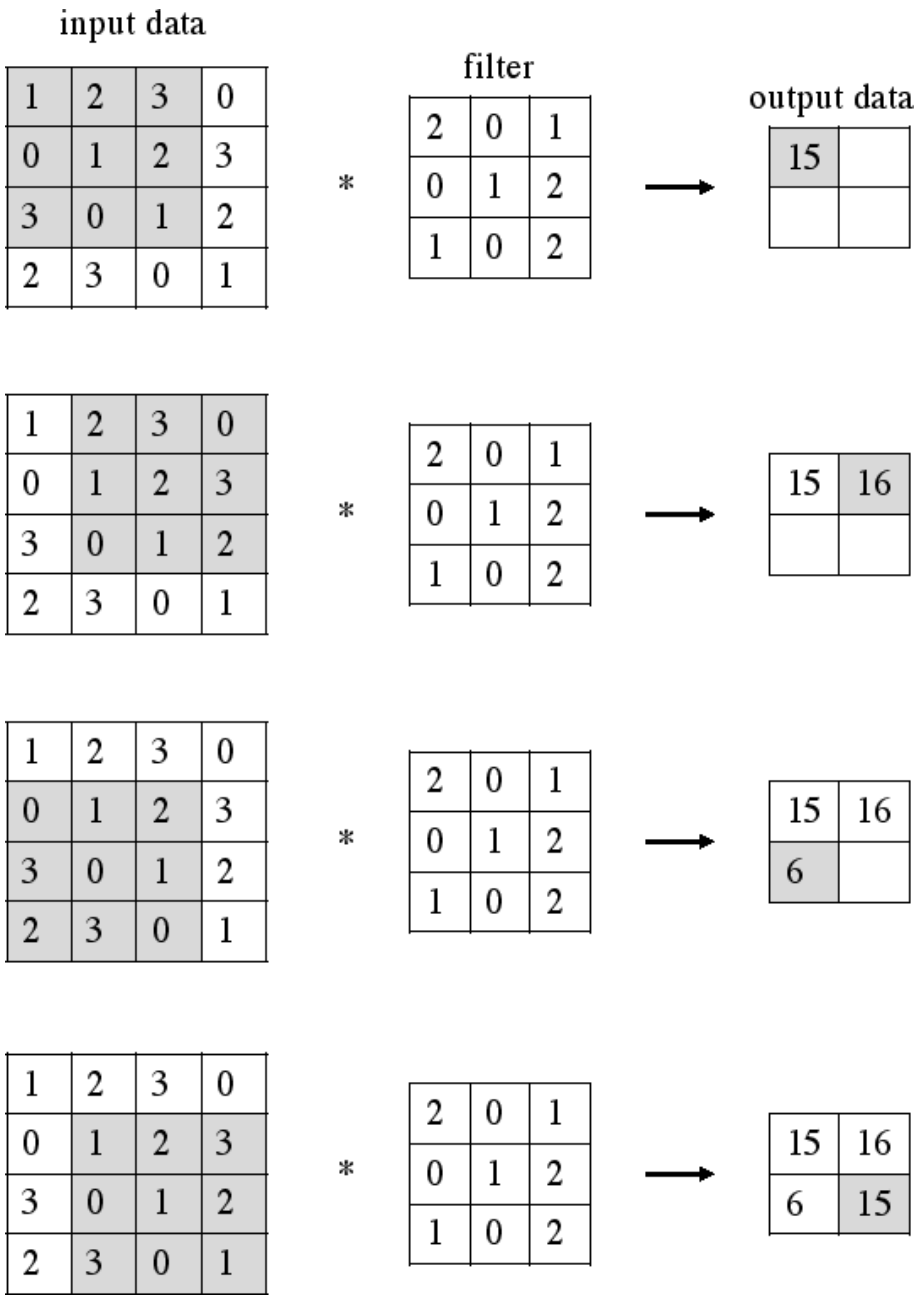

Figure 6.3　Procedure of calculating convolution operation[5)]

横ストライド, 1] の形式で設定する。この場合のストライドは 1×1 となっている。入力サイズが 7×7 のデータにストライド 2×2 を設定した場合の例を Figure6.6 に示す。第 4 引数はパディングであり、畳み込み層の処理を行う前に入力データの周囲に固定のデータを埋める処理のことである。入力サイズが 4×4 のデータに 0 パディングを適用した例を Figure6.6 に示す。[5)]

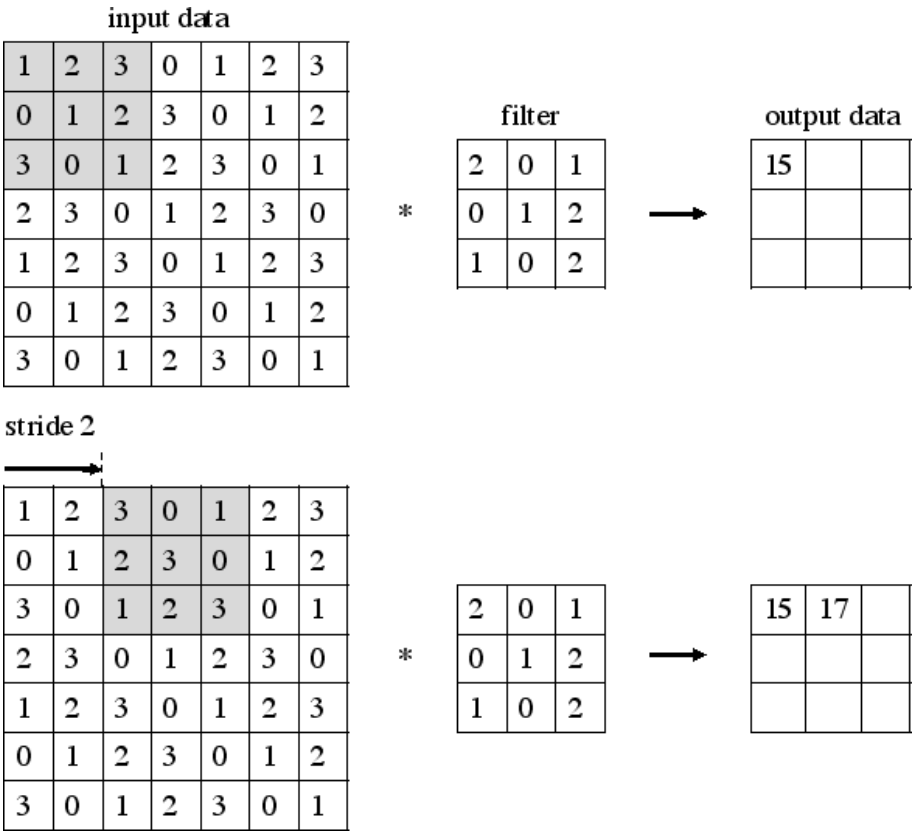

Figure 6.4　Stride[5)]

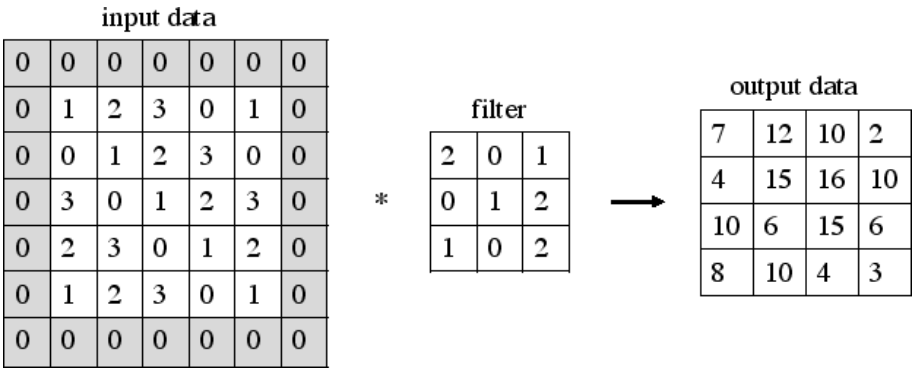

Figure 6.5　Padding[5)]

6.4.4　プーリング層の構築

プーリング層を返すメソッドを作成する。プーリングは入力画像のスケールを小さくする演算である。Figure6.6 に示すように、2×2 の領域を一つの要素に集約するような

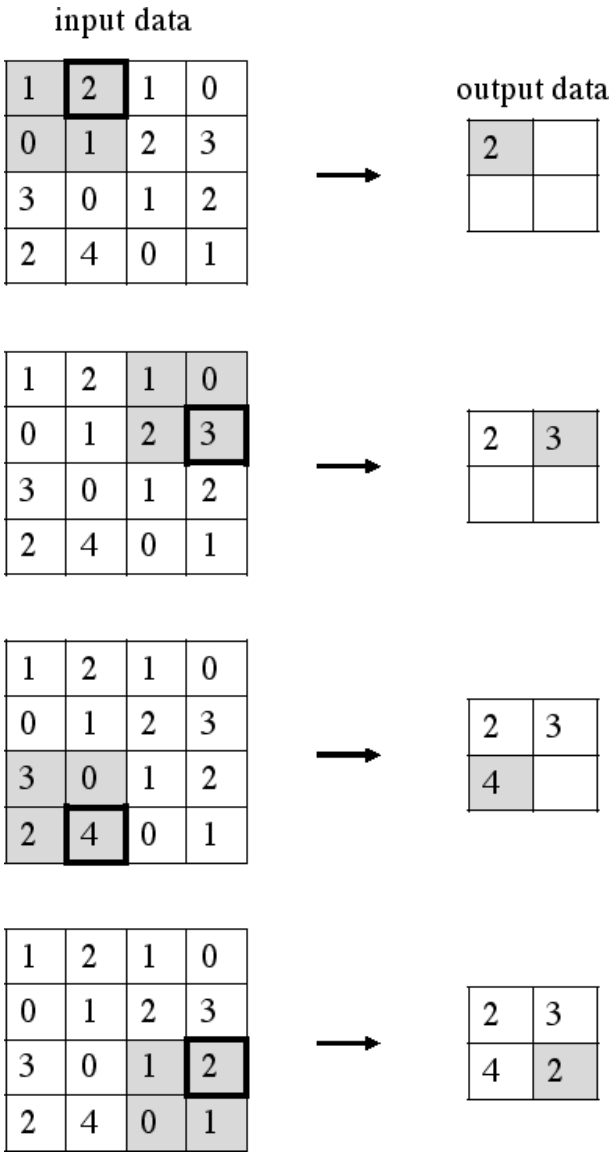

Figure 6.6　Max pooling procedure[5)]

処理を行い、空間サイズを小さくしている。

```
def max_pool_2x2(x, W):
    return tf.nn.max_pool(x, kside=[1, 2, 2, 1],
    strides=[1, 2, 2, 1], padding='SAME')
```

Figure6.6 の例は、2 × 2 の Max プーリングをストライド 2 × 2 で行った場合の処理手順である。Max プーリングは最大値を取る演算であり、2 × 2 とは対象の領域のサイズを表す。ストライドは 2 × 2 に設定しているため、2 × 2 のウィンドウの移動間隔は 2 要素

ごとになる。一般的には、プーリングのウィンドウサイズとストライドは同じ値に設定する。第1引数は入力データであり、畳み込み層からの出力データを示す。第2引数は、プーリングサイズであり、[1, 縦プーリング, 横プーリング, 1] の形式で表す。配列の第0要素と第3要素については、ストライドと同様である。この場合は 2×2 の領域で Max プーリングを行っている。[5)]

6.4.5 プレースホルダの構築

```
x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])
```

プレースホルダとは、計算グラフの実行時に値をフィードするために事前に確保しておく領域を指す。TensorFlow ではデータをテンソルとして扱うが、プレースホルダにフィードするテンソルの型はその構築時に指定しなければならない。MNIST の入力次元は 28×28 の計 784 次元から構成され、出力次元は 0〜9 の計 10 次元から構成されるため、これを受容する形状のプレースホルダを構築する。引数に None を指定すると、そのサイズをセッションの実行時に定めることができる。

6.4.6 第1畳み込み層、プーリング層

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

第1畳み込み層では 5×5 サイズのフィルタを適用して畳み込み処理を行い、1チャンネルの入力画像から 32 チャンネルの特徴を抽出している。ReLU 関数を適用することで、コンピュータに特徴として抽出されない負値を 0 として出力する。[5)]

```
h_pool1 = max_pool_2x2(h_conv1)
```

第1プーリング層では、第1畳み込み層から得られた出力に対しプーリングを行う。第1プーリングを行った時点では、14×14サイズ、32チャンネルのデータとなっている。[5)]

6.4.7 第2畳み込み層、プーリング層

```
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

第2畳み込み層及びプーリング層でも、第1層と同様の処理を行なっている。しかし、第1プーリング層での出力時点で画像が14×14サイズ、32チャンネルのデータとなっているため、第2畳み込み層では32チャンネルから64チャンネルの特徴を抽出している。[5)]

```
h_pool1 = max_pool_2x2(h_conv1)
```

第2プーリング層においても同様であり、第2プーリングを行った時点で、画像は7×7サイズ、64チャンネルのデータとなっている。

6.4.8 第1全結合層

分類問題を解くために、全結合層を挿入し入力画像を最終的に1次元のベクトルに変換する必要がある。全結合層の実装方法を以下に示す。

```
W_fc1 = weight_variable([7*7*64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

6.4.9 ドロップアウト層

```
keep_prob = tf.placeholder(tf.float32)
```

```
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

過学習への対策として、ドロップアウト層を挿入する。過学習とは、訓練データのみに対し過剰に学習したことが原因で、訓練データに含まれないデータに対する識別精度が低下する現象を指す。機械学習の目標は汎化性能であり、あらゆる入力画像に対し正しく識別できるモデルが望まれる。ニューロンを乱択的に消去するドロップアウト層を挿入することによって、この過学習を抑制する効果が期待される。

6.4.10 第 2 全結層（読み出し層）

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

第 1 全結合層で得られたベクトルにソフトマックス関数を適用することで、出力値を確率に変換することができる。これにより、第 2 全結合層では入力データがどの数字に近いか、即ちどの正解ラベルに分類されるかを判別している。

6.4.11 モデルの訓練と評価

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
sess.run(tf.initialize_all_variables())
for i in range(20000):
  batch = mnist.train.next_batch(50)
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
```

損失関数にはクロスエントロピー誤差を、最適化手法には Adam 法を用いる。TensorFlow では、これらの処理を上記のように簡潔に記述することが可能である。tf.train.AdamOptimizer の第 1 引数には、学習率を指定する。

第7章　実験

7.1　設計

制作したAIのクラス図をFigure7.1に示す。

7.1.1　入力データ

Table7.1のような1つの局面は、Table7.2に示すように、何も置かれていないマスを0、黒駒が置かれているマスを1、白駒が置かれているマスを−1に変換し扱う。

7.2　実験1：6x6リバーシAIの作成

7.2.1　教師信号

6x6リバーシの最善手は1993年にイギリスの研究者Joel Feinsteinによって解明されているため、これを教師信号として用いる。お互いが最善手を打ち続けた場合の全ての局面の評価値を一律100点として入力し、学習は10000回行うものとする。学習には入力層、畳み込み層とプーリング層各4層、ドロップアウト層、出力層の全11層で構築された一般的な畳み込みニューラルネットワークを用いる。第一畳み込み層では、$5 \times 5 \times 1$のサイズのフィルタをストライド1で適用し、32chのデータを抽出する。padding引数はsameと設定し、フィルタによってスケールが縮まないようゼロパディングを行う。活性化関数にはReLUを用いる。第一プーリング層では、2×2のサイズ、2×2のストライドでMAXプーリングを行う。以降のプーリング層でも、これと同じ構築で設計する。第二畳み込み層では、$5 \times 5 \times 32$のサイズのフィルタをストライド1で適用し、64chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。第三畳み込み層では、$5 \times 5 \times 64$のサイズのフィルタをストライド1で適用し、256chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。第四畳み込み層では、$5 \times 5 \times 256$のサイズのフィルタをストライド1で適用し、256chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。ドロップアウト層のドロップアウト率は、0.5として実装する。

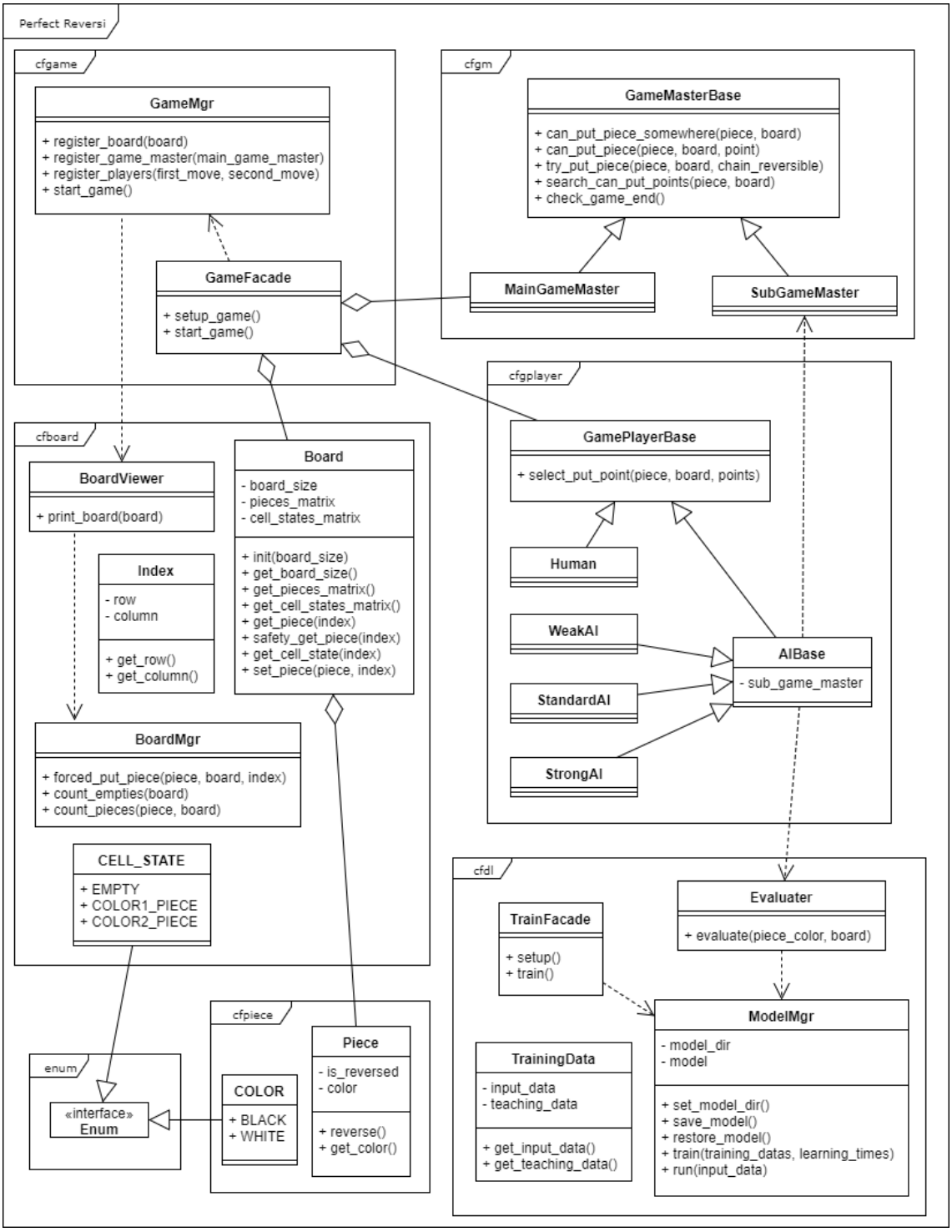

Figure 7.1　Class diagram

7.2.2　実験結果

1000 回対戦させた時の対戦結果を Table7.3 に示す。表中の値は、決着時の石数の平均値である。また、学習時における損失の変化を Figure7.2 に示す。グラフの横軸は学習回数を示しており、縦軸は損失を示している。

Table 7.1　Board state 1

			○						
			○	○		●			
	●	●	○	●	○	●			
			○	●	○	○	○		
				○					
				●	○	●			
						○			

Table 7.2　Board state 2

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	-1	0	0	0	0	0	0
0	0	0	-1	-1	0	1	0	0	0
0	1	1	-1	1	-1	1	0	0	0
0	0	0	-1	1	-1	-1	-1	0	0
0	0	0	0	-1	0	0	0	0	0
0	0	0	0	1	-1	1	0	0	0
0	0	0	0	0	0	-1	0	0	0
0	0	0	0	0	0	0	0	0	0

7.2.3　考察

ランダム同士の対戦結果から、6 × 6 マスのリバーシでは後手が 2 石ほど優位であると考えられる。先手が AI、後手がランダムの対戦結果を見ると、ランダム同士の時よりも約 1 石ほど石数が多くなっている事が分かる。この結果から、お互いが最善手を打ち続

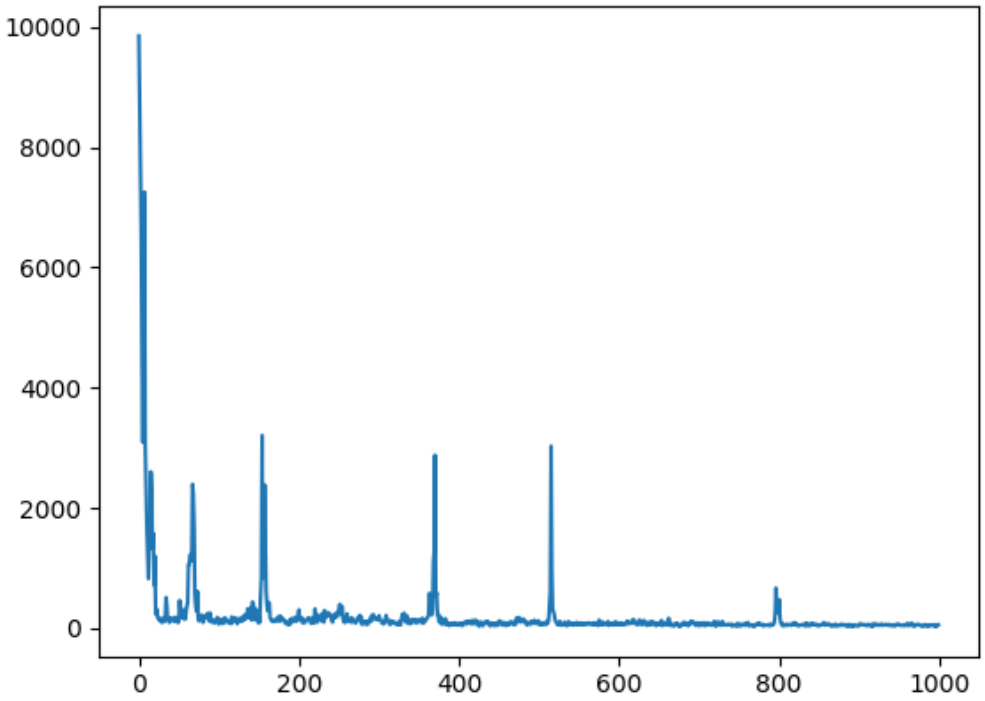

Figure 7.2　Loss - 6x6

Table 7.3　Match result - 6x6

[pieces]		2nd move	
		AI	Random
1st move	AI	-	17.782 / 18.152
	Random	17.222 / 18.735	16.950 / 19.002

けた場合の 1 局分を学習させた場合、AI は先手ならば約 1 石分ほど強化されると考えられる。先手がランダム、後手が AI の対戦結果を見ると、ランダム同士の時よりも約 0.2 石ほど石数が少なくなっていることが分かる。この結果から、今回の実験で用いた学習手法では後手番の場合僅かに弱化してしまうと考えられる。この原因として、以下の 3 つの可能性が考えられる。

- 6×6 マスのリバーシの学習において、今回の実験で構築した畳み込みニューラルネットワークが適切な構築ではなかった
- 教師信号として与えた評価値が適切な値ではなかった

- 訓練データセットの数が学習に十分ではなかった

7.3 実験2：パーフェクト・リバーシAIの作成

7.3.1 教師信号

最善手が判明している6×6のリバーシとは異なり、パーフェクト・リバーシAIの学習に用いる訓練データには様々なものが候補として挙げられる。今回の実験では、「角を取る手が有力手である可能性が高い」という仮説に基づき、それらの手を打った局面と完封した局面を評価値1000点として入力する。学習は実験1と同様、10000回行うものとする。学習には入力層、畳み込み層とプーリング層各5層、ドロップアウト層、出力層の全13層で構築された一般的な畳み込みニューラルネットワークを用いる。第一畳み込み層では、5×5×1のサイズのフィルタをストライド1で適用し、32chのデータを抽出する。padding引数はsameと設定し、フィルタによってスケールが縮まないようゼロパディングを行う。活性化関数にはReLUを用いる。第一プーリング層では、2×2のサイズ、2×2のストライドでMAXプーリングを行う。以降のプーリング層でも、これと同じ構築で設計する。第二畳み込み層では、5×5×32のサイズのフィルタをストライド1で適用し、64chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。第三畳み込み層では、5×5×64のサイズのフィルタをストライド1で適用し、256chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。第四畳み込み層では、5×5×256のサイズのフィルタをストライド1で適用し、256chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。第五畳み込み層では、5×5×256のサイズのフィルタをストライド1で適用し、256chのデータを抽出する。パディング及び活性化関数は第一畳み込み層と同様に設定する。ドロップアウト層のドロップアウト率は、0.5として実装する。

7.3.2 実験結果

1000回対戦させた時の対戦結果をTable7.4に示す。表中の値は、決着時の石数の平均値である。また、学習時における損失の変化をFigure7.3に示す。グラフの横軸は学習回数を示しており、縦軸は損失を示している。

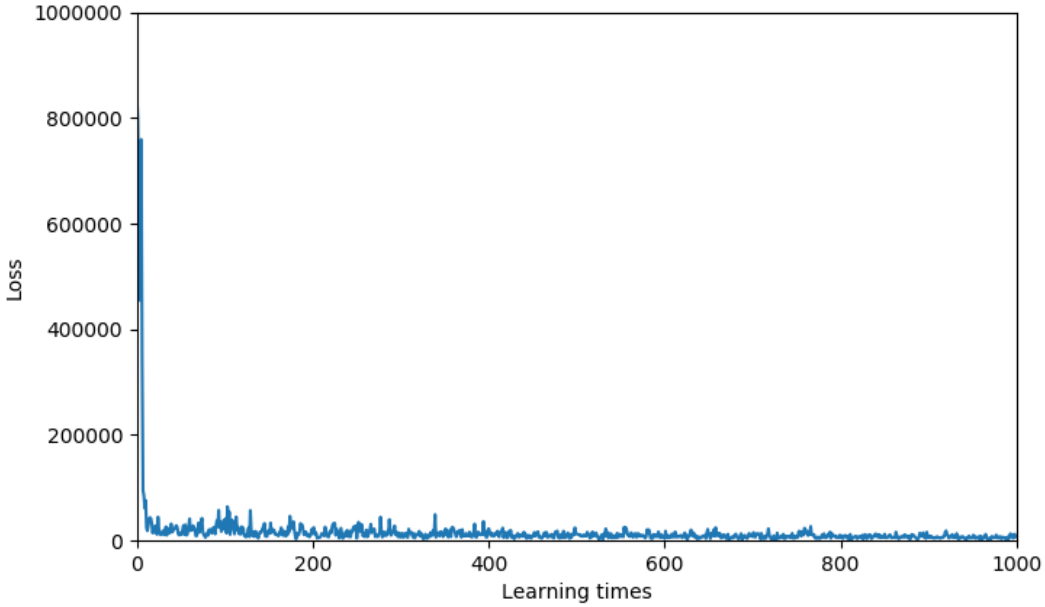

Figure 7.3　Loss - 10x10

Table 7.4　Match result - 10x10

[pieces]		2nd move	
		AI	Random
1st move	AI	-	49.216 / 50.555
	Random	49.443 / 50.555	49.894 / 50.103

7.3.3　考察

ランダム同士の対戦結果から、10×10 マスのリバーシでは後手が約 0.2 石ほど優位であると考えられる。先手が AI、後手がランダムの対戦結果を見ると、ランダム同士の時よりも約 0.7 石ほど石数が少なくなっている事が分かる。この結果から、今回の実験で用いた学習手法では先手番の場合僅かに弱化してしまうと考えられる。先手がランダム、

後手がAIの対戦結果を見ると、ランダム同士の時よりも約0.5石分ほど石数が多くなっている事が分かる。この結果から、後手番の場合は今回の実験で用いた学習手法で約0.5石分ほどAIが強化されると考えられる。先手番で石数が少なくなった原因として、以下の3つの可能性が考えられる。

- 10×10 マスのリバーシの学習において、今回の実験で構築した畳み込みニューラルネットワークが適切な構築ではなかった
- 教師信号として与えた評価値が適切な値ではなかった
- 訓練データセットの数が学習に十分ではなかった

特に、10×10 マスのリバーシにおいては通常のリバーシと比べて明確に優勢と判断できる局面が少なく、教師信号には工夫が必要であると考える。

第8章　結論

本研究では、パーフェクト・リバーシと呼ばれる 10×10 マスの大盤リバーシ AI に対し、畳み込みニューラルネットワークを用いて学習を行うことを試みた。また、パーフェクト・リバーシに加えて 6×6 マスの小盤リバーシ AI を制作し、数少ない学習データからどれ程の性能向上を見込めるかについての検証を行った。結果として、6×6 マスのリバーシにおいては若干の性能向上が見られたが、パーフェクト・リバーシにおいては性能の向上が余り見られなかった。学習回数 1 万回時点において、学習した手自体は指していることから、学習そのものは効率的に行うことができていると考えられるが、いずれの AI においても強さという点では目覚ましい性能向上が見られず、今回の実験で用いた訓練データでは学習に不適であるという結論に至った。

今回の実験で用いたニューラルネットワークは、何れも 10 層を超える非常に深いニューラルネットワークであったが、このような深層ニューラルネットワークは、更にサイズの大きいリバーシ、そして同様の手法で学習可能なチェスや囲碁において効果的であると予想される。ただし、教師信号には工夫が必要で、各局面の評価値の定め方の是非は一概に決定することはできない。AI の対戦結果を見ると、教師信号によっては先後で優劣が逆転するなど、評価値の決定の困難性を示している。その性質は特にパーフェクト・リバーシのように盤が大きく、不確定要素が多くなってしまう場合に顕著で、序盤と終盤で形勢が逆転する可能性が高くなるため、評価値の決定がより困難となる。今回の実験で用いた評価値の定め方以外では、リバーシの開放度理論を用いた定め方などが候補として挙げられる。

今回の実験では、ニューラルネットワークの構築に TensorFlow ライブラリを用いたが、TensorFlow は演算や最適化などの内部的な処理を気にせずにソースコードを記述する事ができるため、実装上のバグが生じにくいという利点がある。TensorFlow には TensorBoard という計算グラフを可視化する機能も搭載されており、不具合を可視化できるという点も、バグを生じにくくさせる要因の 1 つとなっている。

TensorFlow を始めとして、Chainer や Keras など、今日では数多くの機械学習向けライブラリがするが、これらのような強力なサポートツールがより一層世間に浸透し、科学技術の発展はもちろんのこと、企業や教育といった方面でも幅広く活用されていくことを願いたい。

8.1　謝辞

本研究を進めるにあたり、御多忙中にも関わらず多大なご指導を賜りました出口利憲先生に深く感謝すると共に、同研究室において共に勉学に励んだ久保田勘太郎氏、稲垣天斗氏、竹中一生氏に厚く御礼を申し上げます。

参考文献

1) 新井 康允：脳とニューロンの科学, 裳華房（2000）

2) 臼井 支郎：基礎と実践 ニューラルネットワーク, コロナ社（1995）

3) 中居 悦司：TensorFlow で学ぶディープラーニング入門〜畳み込みニューラルネットワーク徹底解説, マイナビ出版（2017）

4) 斎藤 康毅：ゼロから作る Deep Learning Python で学ぶディープラーニングの理論と実装, オライリー・ジャパン（2017）

5) 後藤 匠：深層学習を用いたゲームの局面評価の研究, 岐阜工業高等専門学校電気情報工学科卒業研究報告（2017）

6) TensorFlow,https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners,2017 年 2 月 16 日アクセス

7) 兼松 良輔：ニューラルネットワークによるゲームの局面評価の研究, 岐阜工業高等専門学校電気情報工学科卒業研究報告（2014）