卒 業 研 究 報 告 題 目

CNNにおけるネットワーク構成に関する研究

Study of Evaluation Function for Game 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

2014E07 今井 恭平

平 成 ３ １ 年（ ２ ０ １ ９ 年 ）　　２ 月 １ ７ 日 提 出

Abstract

In deep learning, a Convolution Neural Network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. The purpose of this study is to create a convolution neural network which is higher accuracy than previous one. Improvement of the accuracy is one of the most important elements for the applicability to the real world. In this study, it learns by the existing game record data of shogi, and implements the AI which moves the best way with Chainer. In addition, Residual Network which is the method to enable CNN to become deep is implemented and it's compared and examined about the improvement of the accuracy. As a result, when the number of layers were increased, the accuracy was found falling. But I succeed to plan for improvement of the accuracy by Residual Network (ResNet) which used a residual. One of the problem is that the training time becomes long. Reducing training time is task to be solved.

目 次

第1章　Introduction

現代の生活において人工知能 (AI) は切り離せないものとなっており、現在でも急速な発展を続けている。その応用範囲は、自然言語処理や画像認識など多岐にわたる。その中でも深層学習と呼ばれる層の深いものは複雑な学習を行える為、近年特に注目を浴びている。また、自動運転技術の開発など命に関わる場面でも応用されることが期待されている。その為にニューラルネットワークの学習精度を高くするということが求められる。本研究では、将棋と呼ばれるゲームの既存の棋譜データによって学習を行い、最善手を打つ AI を畳み込みニューラルネットワーク (CNN) を Chainer を用いて実装する。このとき CNN における層数やフィルタ数による精度の変化について研究を行う。さらに、CNN の深層化を可能にする手法である Residual Network を実装し、精度の向上手法について比較検討を行う。

第2章　Neural Network

2.1　Neuron

ニューロン (Neuron) とは、Figure 2.1 に示すような生体の神経系を構成する神経細胞である。これは情報処理に特化しているという特徴を持つ。ニューロンは細胞体と樹状突起、軸索に分けることができる。細胞の中央部分にあたるのが細胞体であり、細胞核はこの中にある。 タンパク質など細胞の活動に必要な物質は細胞体の中で合成され、 特殊な輸送機構により細胞内のすみずみまで送られる。 樹状突起は、細胞体の表面から突き出た多くの枝分かれをもった突起を指していう。ニューロンは神経細胞へ入力刺激が入ってきた際、活動電位を発生させ、他の細胞に情報を伝達する働きがある。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、軸索によって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。

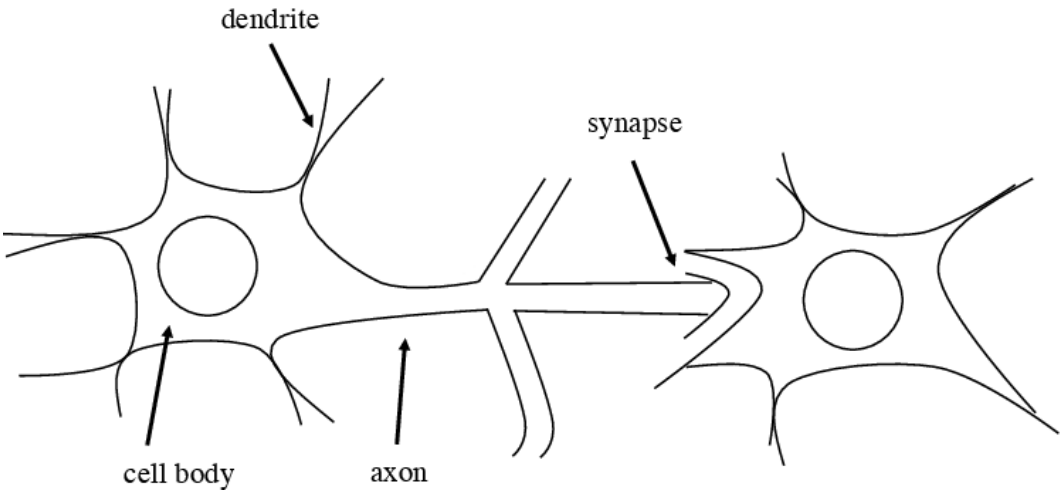

Figure 2.1　Neuron[1)]

2.2 Neuron Model

脳にはいくつもの種類のニューロンが存在することが確認されており、単純な発火作用のみを持ったものから非常に複雑な発火作用を持った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に用いられるニューロンモデルはFigure 2.2 のように単純化された多入力1出力のものである。入力 x_i に対する結合荷重を w_i とすると、ニューロンの出力 u は式（2.1）で表される。

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

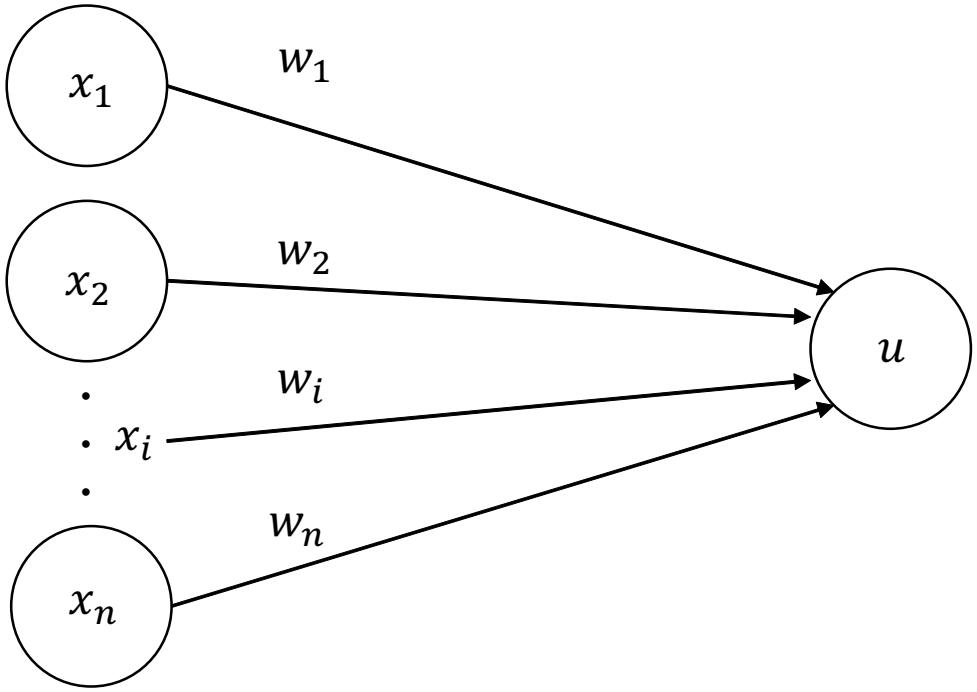

Figure 2.2　Neuron model

2.3 Activation Function

活性化関数 (Activation Function) とはニューラルネットワークにおいて次の層に渡す値を整えるような役割を持つ。この関数は非線形関数若しくは恒等関数である。活性化関数として用いられる関数は様々である。Figure 2.3 は、階段関数、ステップ関数（Step function）と呼ばれるもので、この関数を用いればニューロンの出力を 0 か 1 で表すことができる。ステップ関数は、式（2.2）で表される。

$$f(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{2.2}$$

Figure 2.4 は、シグモイド関数（Sigmoid function）と呼ばれるもので、ステップ関数と比較すると入力に対して連続的に出力が変化する。これを用いることにより、ニューロンの出力を連続的な実数値の信号として表すことができる。シグモイド関数は、式（2.3）で表される。

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2.3}$$

近年では Figure 2.5 に示すような ReLU 関数（Rectified Linear Unit function）と呼ばれる関数がよく用いられる。この関数は、入力が 0 を超えていれば、その入力をそのまま出力し、0 以下ならば 0 を出力する。ReLU 関数は、式（2.4）で表される。

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{2.4}$$

これらの関数は一般的に、入力データから、連続的な数値の予測を行うような回帰問題を解決する際に用いられる。一方、データがどのクラスに属するかを判断するような分類問題を行う際には、Figure 2.6 のようなソフトマックス関数 (Softmax function) が用いられる。出力層を n 個とし、入力信号 a_k から k 番目の出力 y_k を求める式を式（2.5）に示す。

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} \tag{2.5}$$

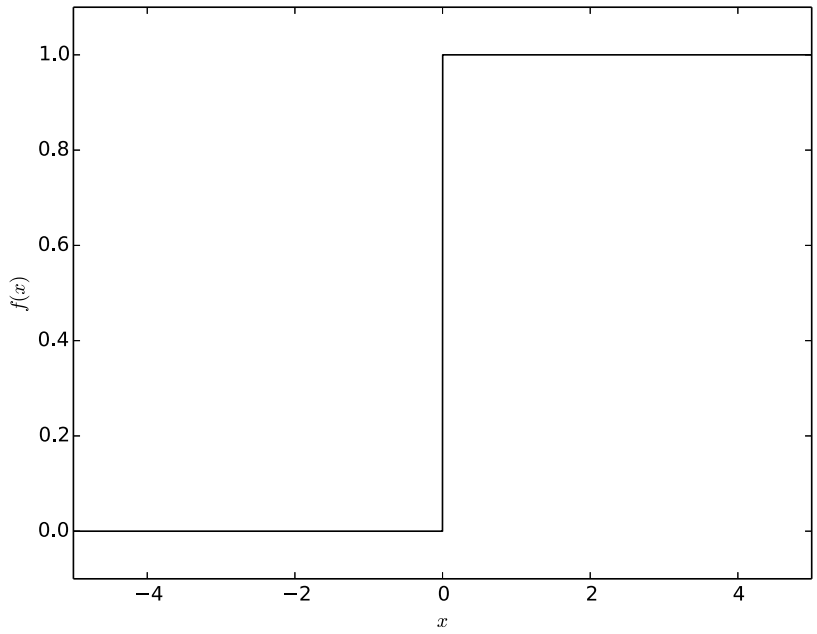

Figure 2.3　Step function

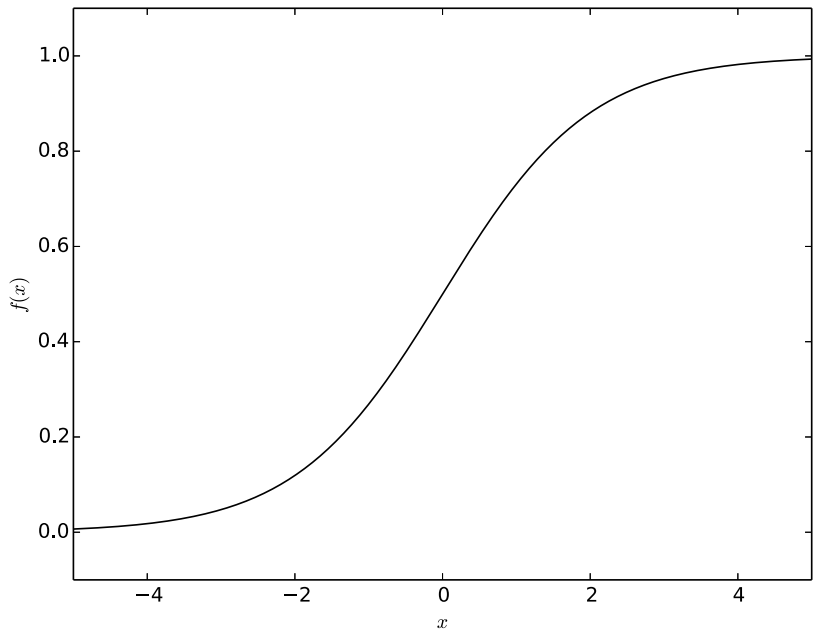

Figure 2.4　Sigmoid function

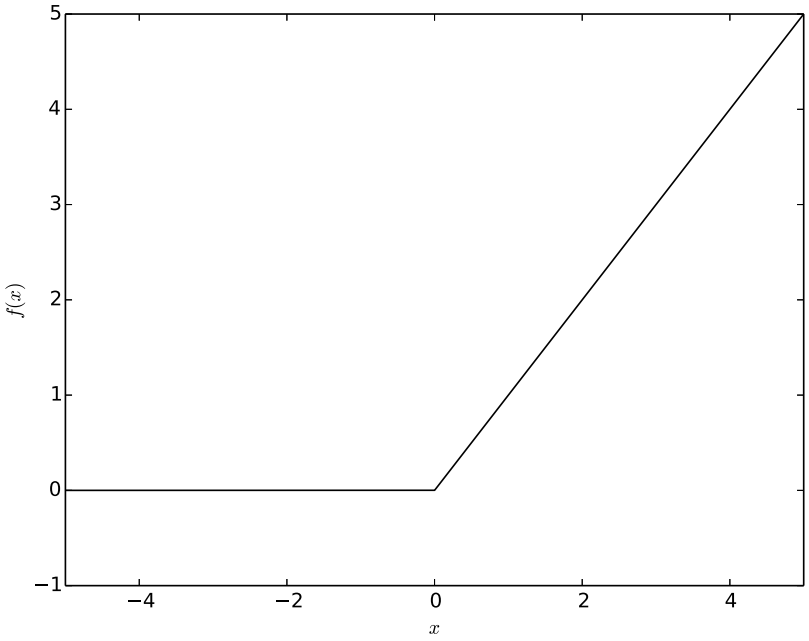

Figure 2.5　ReLU function

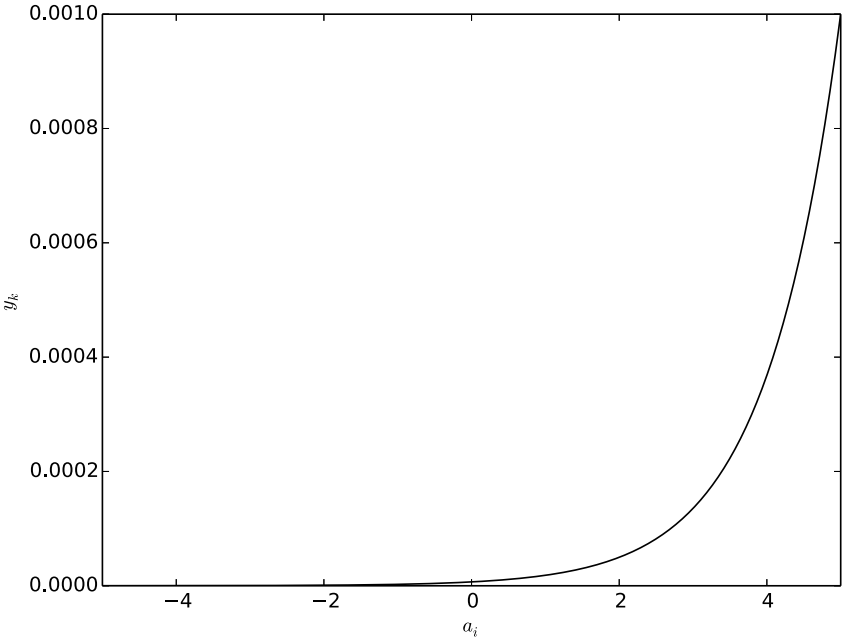

Figure 2.6　Softmax function

第3章　Perceptron

パーセプトロン (Perceptron) は 1957 年にアメリカの心理学者 Frank Rosenblatt が考案したニューラルネットワークの一種である。シンプルな構造でありながら学習力を持つため、1960 年代に爆発的なブームとなった。しかし、1969 年にアメリカの人工知能学者 Marvin Minsky らによって単層のものは線形分離可能なものしか学習できないことが指摘され下火となった。パーセプトロンは AND ゲートや OR ゲートなどの論理回路を表すことができるが、XOR ゲートなどの論理回路は表すことができない。これの例を Figure 3.1 に示す。その後、1980 年代中盤に Boltzmann machine や、Backpropagation などの手法が開発されたため、再び注目を集めた。[2)]

3.1　Single-layer Perceptron

単純パーセプトロン (Single-layer Perceptron) は Figure 3.2 のようなユニットで構成される。また、1 つのユニットは、次の要素で構成される。

- 入力 x_1, x_2, ..., x_n
- それぞれの入力に対応した重み w_1, w_2, ..., w_n
- 活性化関数 f
- 1 つの出力 z

入力 x_1, x_2, ... , x_n はニューロンに伝達される信号を表し、重み w_1, w_2, ..., w_n はニューロンの結合の強度を表している。ユニットへの入力は、式（3.1）で表される。

$$u = \sum_{i=1}^{n} x_i w_i \tag{3.1}$$

出力 z は式（3.2）で表される。

$$z = f(u) \tag{3.2}$$

パーセプトロンでは活性化関数として古くからシグモイド関数が使われている。シグモイド関数はしきい値が 0.5 と固定されているため、しきい値を調整するため、式（3.1）に値 b を加え、式（3.3）のように変形する。

$$u = \sum_{i=1}^{n} x_i w_i + b \tag{3.3}$$

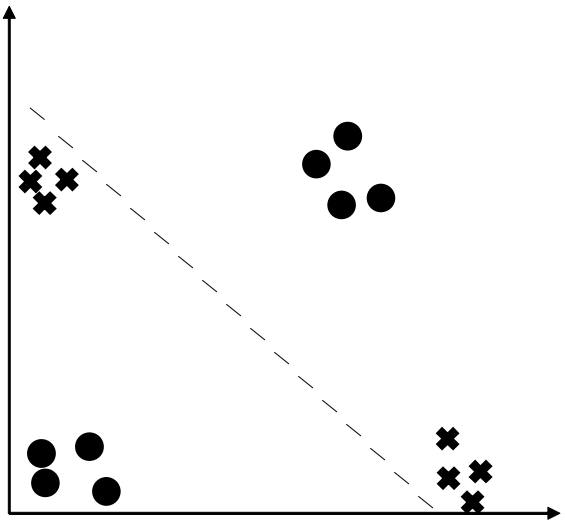

Figure 3.1　Limit of perceptron

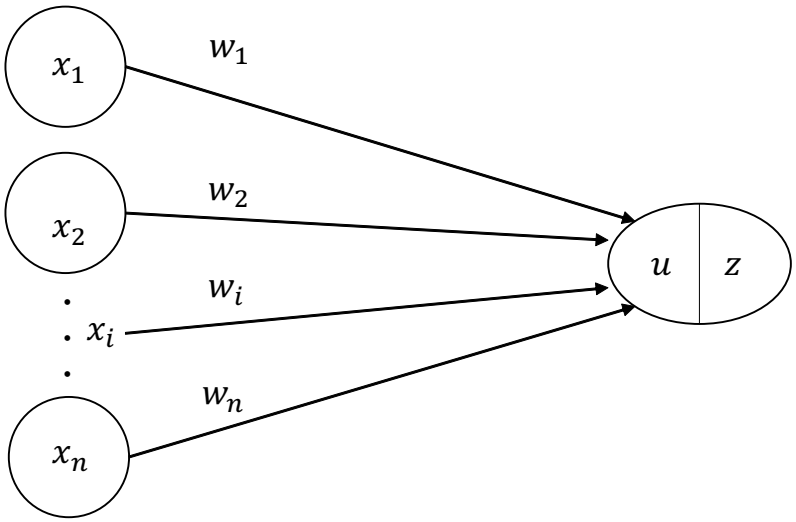

Figure 3.2　Perceptron

この加えた値をバイアスと呼び、活性化関数のしきい値を調整する役割を持つ。

3.2 Multi-layer Perceptron

パーセプトロンのユニットを多層に連結したニューラルネットワークは、多層パーセプトロン (Multi-layer Perceptron) と呼ばれる。多層パーセプトロンの入力と出力に当たる層をそれぞれ入力層、出力層と呼び、それ以外を中間層と呼ぶ。入力層、中間層、出力層の 3 層からなる多層パーセプトロンは Figure 3.3 のようになる。

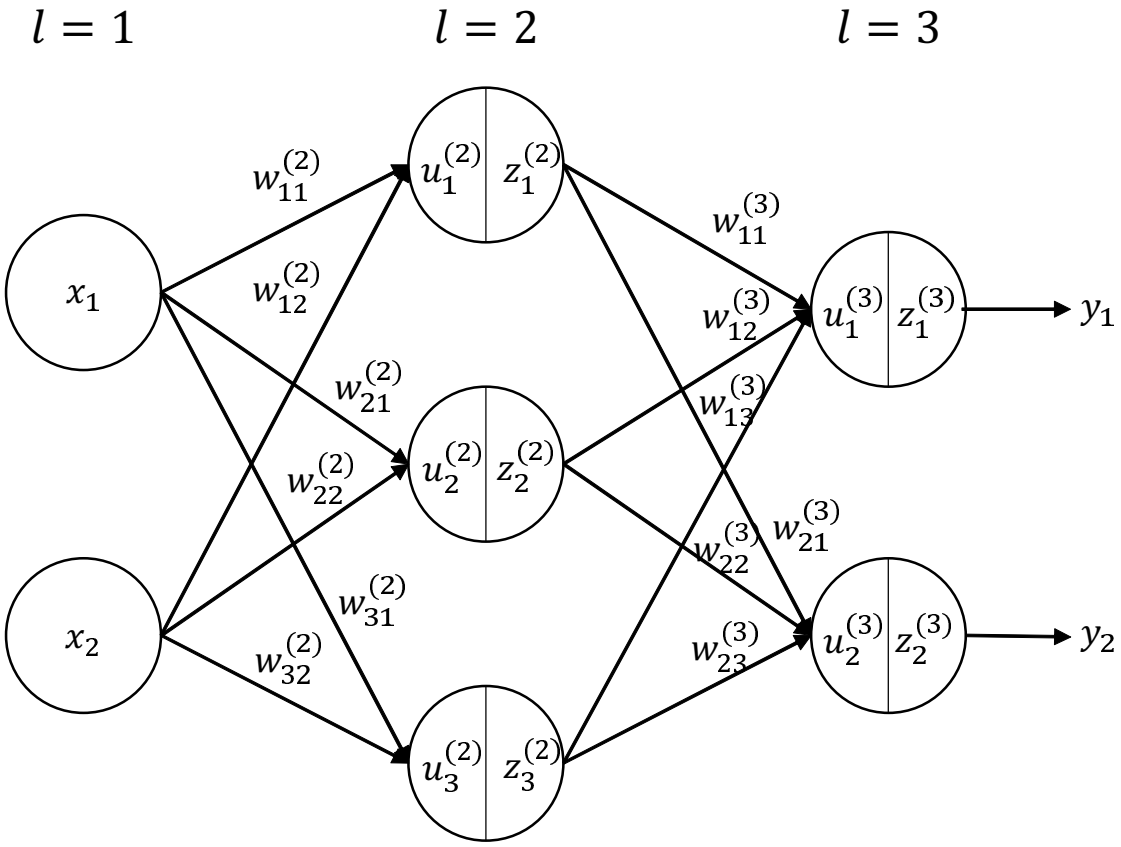

Figure 3.3　Multi-layer perceptron

ユニットの連結は重み $w_{ji}^{(l)}$ を持ち、添字は $l-1$ 層の i 番目のユニットと l 層の j 番目のユニットの連結であることを示し、重みはニューロンの結合強度を表す。l 層の j 番目のユニットへの入力は、$l-1$ 層のユニットが I 個であるとすると式（3.4）で表される。ただし、各ユニットはバイアス $b_i^{(l)}$ を持つとする。

$$u_j^{(l)} = \sum_{i=1}^{I} w_{ji}^{(l)} z_i^{(l-1)} + b_j^{(l)} \tag{3.4}$$

重み $w_{ji}^{(l)}$ やバイアス $b_i^{(l)}$ のことをニューラルネットワークのパラメータという。l 層の j 番目のユニットの出力 $z_j^{(l)}$ は、活性化関数を f とすると、式（3.5）で表される。

$$z_j^{(l)} = f(u_j^{(l)}) \tag{3.5}$$

入力層は、入力値 x_j がそのまま出力 $z_j^{(1)}$ となるため、式（3.6) で表される。

$$x_j = z_j^{(1)} \tag{3.6}$$

出力値 y_j は、出力層ユニットの出力となり、ニューラルネットワークの層数を L 層とすると、式（3.7）で表される。

$$y_j = z_j^{(L)} \tag{3.7}$$

中間層の活性化関数には古くからシグモイド関数が使用されていたが、シグモイド関数は、原点から遠ざかるほど勾配が 0 に近づくため、学習が進み活性化関数への入力値が大きな値になると、ユニットの学習が停滞するという問題があった。これは勾配消失問題と呼ばれる。勾配消失問題を解決するために、中間層の活性化関数には原点から離れても勾配が変わらない ReLU 関数が使われる。

第4章　Deep Learning

ディープラーニング (Deep Learning) とは多層化し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層以上のニューラルネットワークには勾配消失問題や過学習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、過学習を防止するドロップアウトの発案や、ReLU 関数の発案、計算機の性能向上などの理由により、2010 年台にディープニューラルネットワークを用いた学習が可能となった。ディープラーニングは特に画像認識や音声認識を行う際、非常に強力なツールとなる。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。

- Convolution Neural Network
- Residual Network
- Stacked Auto Encoder
- Deep Belief Network
- Recurrent Neural Network

4.1　Convolution Neural Network

ニューラルネットワークで画像や音声を対象とする場合、畳み込みニューラルネットワーク (Convolution Neural Network) が使われる。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。畳み込みニューラルネットワークの構築例を Figure 4.1 に示す。全結合層では隣接する層におけるすべてのニューロン間で結合がある層を示す。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。

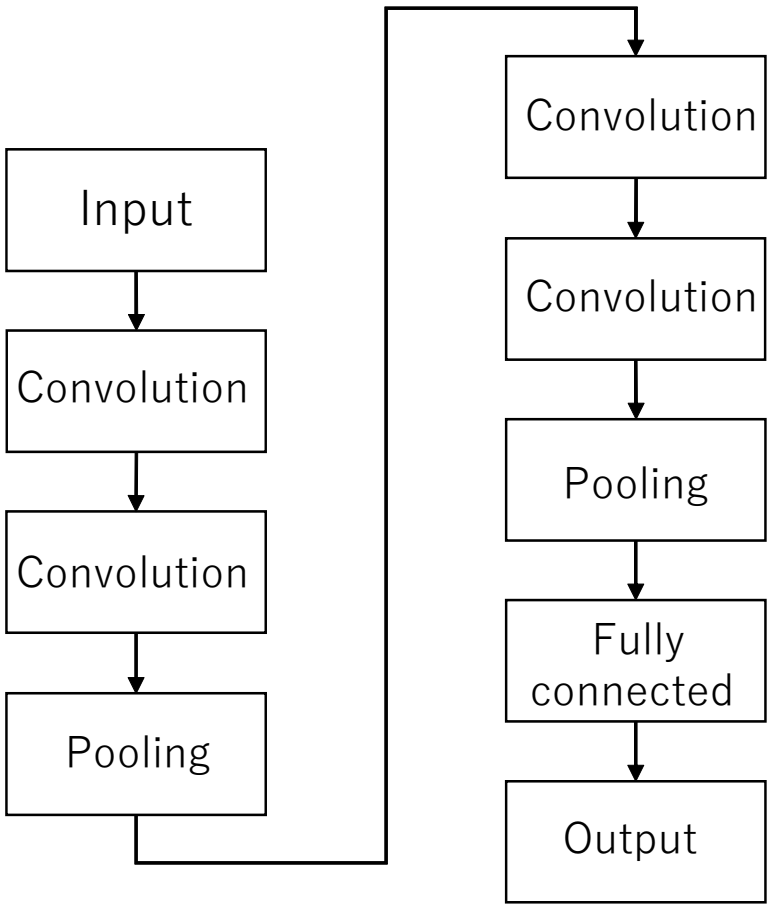

Figure 4.1　Convolution neural network

4.2　Residual Network

Residual Network(ResNet) は Microsoft Research の Kaiming He 氏らが 2015 年に考案したニューラルネットワークのモデルである。[3] 画像分類問題において、ネットワークの層の深さは重要であり、深いほど精度向上すると考えられているが、より深いネットワークを学習させようとすると、Figure 4.2 のように精度が劣化することが知られている。ResNet はこのような深いネットワークにおいて発生する精度が劣化する問題を解決することができる。

ResNet では直接最適な写像（変換）になるよう学習するのではなく、残差の写像が最適になるよう学習を行う。求める写像を $\mathcal{H}(x)$ とすると、入力 x との残差 $\mathcal{F}(x)$ は $\mathcal{H}(x)-x$ で表すことができ、元の写像は $\mathcal{F}(x)+x$ となる。元の写像 $\mathcal{F}(x)+x$ は Figure 4.3 のようにショートカット接続により実現できる。ショートカット接続はいくつかの層をスキップする単なる恒等写像であるが、パラメータの追加がなく、計算も複雑にならず、逆誤差伝播も可能であるため実装も容易といったメリットがある。

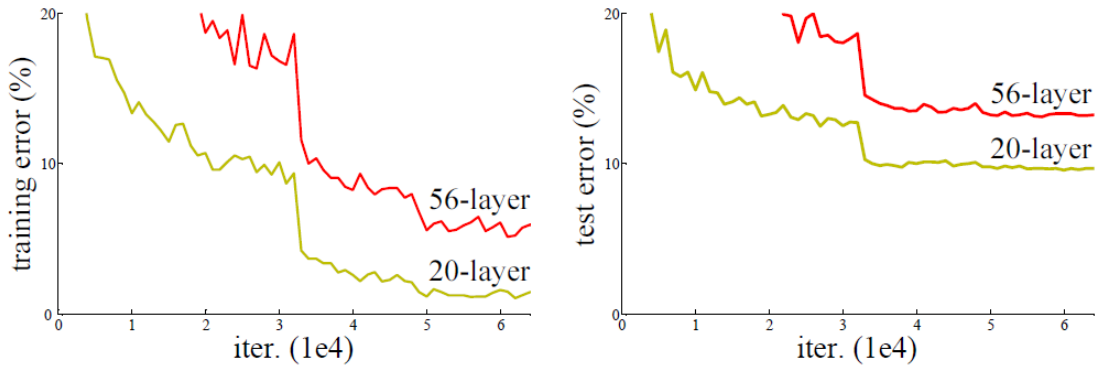

Figure 4.2　Training error of deeper network[3)]

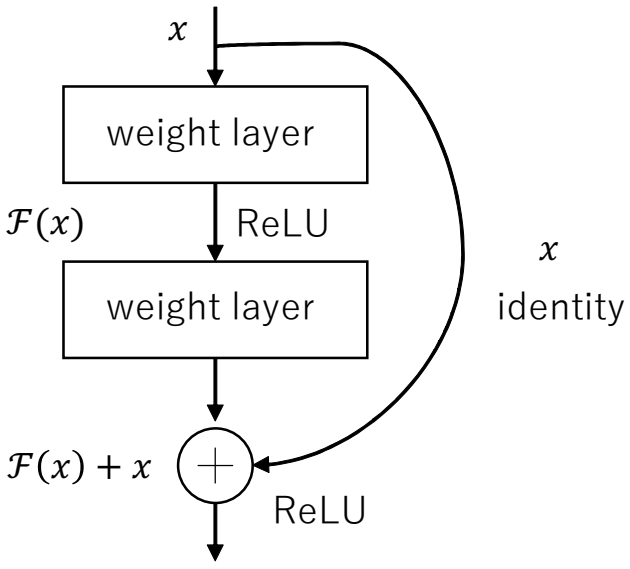

Figure 4.3　Short cut connection[3)]

第5章　学習方法

5.1　誤差関数

入力と正解となる出力のペアを与えてニューラルネットワークを学習することを、教師あり学習と呼び、あらかじめ与えられる例題と答えについてのデータを学習データと呼ぶ。ニューラルネットワークの出力と教師データとの誤差の尺度を損失という。損失の値が大きいほど正解からは離れているということを意味する。この損失を評価する関数を誤差関数と呼ぶ。出力層の活性化関数によって Table 5.1 の誤差関数がよく使われる。誤差関数には次の特徴がある。

- スカラー値を出力とする
- 単調増加関数である
- 微分可能である

Table 5.1　Error function

activation function	error function	formula
sigmoid function	cross-entropy error	$-t \log y - (1-t) \log (1-y)$
softmax function	cross-entropy error	$-\sum_{k=1}^{K} t_k \log y_k$
ReLU function	mean squared error	$(y-t)^2$

上記の式において K は出力層の合計、 y_k は活性化関数の k 番目の出力、t_k は教師データの k 番目のサンプルを表す。ソフトマックス関数の場合、出力は多値となる。訓練データは、正解ラベルのユニットのみ 1 となり、他のユニットは 0 となる one-hot ベクトルとなる。

5.2　誤差逆伝播法

損失関数の最小化は、勾配降下法を使って行う事ができる。[4] 勾配降下法では、損失関数を結合荷重で偏微分した勾配を求める。損失関数を E 、結合荷重を $\boldsymbol{W}$ とすると、勾配 ∇E は式（5.1）で表される。

$$\nabla E = \frac{\partial E}{\partial \boldsymbol{W}} \tag{5.1}$$

この勾配に学習率を乗じた値で、結合荷重を更新する。結合荷重の更新量 $\Delta \boldsymbol{W}$ は学習率を η とすると式（5.2）で表される。

$$\Delta \boldsymbol{W} = -\eta \nabla E \tag{5.2}$$

学習率 η は、値が大きいほど早く学習が進むが、損失関数の極小点周辺では極小点を通り過ぎてしまう。また、値が小さい場合、極小点へ近づくまでの時間が長くなる。そのため、学習が進むにつれて徐々に値を小さくするのが一般的である。また、他の手法と比較すると局所的最小値に陥る可能性が高い。

勾配 ∇E を式（5.1）に従って計算すると、各層の活性化関数を階層的に呼び出された式の偏微分を計算する必要がある。その際、各結合荷重について計算すると計算量が非常に多くなる。そこで効率的に計算を行うため、誤差逆伝播法 (back propagation) という手法が用いられる。誤差逆伝播法では、偏微分におけるチェインルールを用いることで、出力層から入力層に向かって、連鎖的に結合比重の勾配を計算することができる。損失関数を E 、第 l 層の i 番目のユニットの出力を $z_i^{(l)}$ とすると L 層パーセプトロンの損失関数の偏微分は式（5.3)、式（5.4)、式（5.5）で表される。

$$\frac{\partial E}{\partial W_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)} \tag{5.3}$$

$$\delta_j^{(l)} = \sum_k \delta_{kj}^{(l+1)} \frac{\partial f^{(l)}(u_j^{(l)})}{\partial u_j^{(l)}} \tag{5.4}$$

$$\delta_j^{(L)} = \frac{\partial E}{\partial u_j^{(L)}} \tag{5.5}$$

出力層から入力層に向かって順番に $\delta_j^{(l)}$ を計算することで各層の結合荷重の勾配を求めることができる。

5.3 最急降下法

最急降下法とは最適化問題を解くための勾配法に関するアルゴリズムである。実際のニューラルネットワークでは誤差関数を定義し、これが小さくなるように学習を行う。この誤差を小さくするように各層の重みを更新していくことを学習フェーズで行っている。重み $\boldsymbol{w}$ の更新は式（5.6）のように行われる。

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \frac{\partial E(\boldsymbol{w}^t)}{\partial \boldsymbol{w}^t} \tag{5.6}$$

学習係数 η を恣意的に決定する必要があり、一度決めた η を用いて誤差の最小化を行なっていく。このことから、学習モデルによって最適な値を決めることが難しいという問題がある。Chainer ではデフォルトの値は $\eta = 0.01$ となっている。

5.4 確率的勾配降下法

確率的勾配降下法 (Stochatic Gradient Descent, SGD) とは、最急降下法と同様に最適化問題を解くための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。[5] 確率的勾配降下法では、局所的最小値に収束する可能性が比較的低い。しかし、入力値に平均や分散が極端に偏っている場合、探索が困難な場合がある。

5.5 Adam 法

Adam(Adaptive moment estimation) 法は 2015 年に Diederik P. Kingma らが提唱した手法で、既存の AdaGrad 法や RMSProp 法、AdaDelta 法を改良したものであり、効率的にパラメータを空間を探索することができる。[6]Adam 法には勾配降下法よりも精度が高く収束速度が速いという特徴がある。タイムステップ t におけるパラメータ θ_t に関する目的関数 (誤差関数) の勾配を g_t とすると、Adam 法のパラメータの更新は式（5.7）および式（5.8）のように行われる。[7]

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{5.7}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{5.8}$$

m_t と v_t はそれぞれ、勾配の一次モーメント（平均値）と二次モーメント（分散した平方偏差）の概算値である。m_t と v_t が 0 のベクトルとして初期化されているように、Adam 法の論文では、0 に偏らせることについて述べられている。特に、初期化時点のタイムス

テップと、減衰率が小さくなったとき（つまり β_1 と β_2 が 1 に近づいたとき）である。一次モーメントと二次モーメントの偏りをバイアス補正した推定値を計算することによって、このような偏りを小さくした式は式（5.9）および式（5.10）となる。

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{5.9}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{5.10}$$

これらによって Adam 法の計算式は式（5.11）と導かれる。

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{5.11}$$

論文では、β_1 の初期値は 0.9、β_2 には 0.999、ϵ には 10^{-8} を設定することが推奨されている。

5.6 過学習問題

過学習とは、訓練データに対しは精度が高いが、訓練データにない未知のデータでは予測精度が低くなることをいう。これを防止するため、ドロップアウトという手法が存在する。

5.6.1 ドロップアウト

ドロップアウトは Figure 5.1 のように学習時にニューラルネットワークのユニットの一部をランダムに無効にして学習を行う手法である。独立に訓練した複数のネットワークを使って推論を行い、それらの平均をとることで、個々のネットワークで推論を行うよりも精度が向上することが知られている。ユニットを無効にする割合は 0.5 程度の値を用いることが多い。

5.7 分類問題

5.7.1 2 値分類問題

2 値分類問題は、Figure 5.2 のように入力が 1 つのみで、出力の値が 2 種類に分かれる問題のことである。2 値分類問題では、出力層のユニット数は 1 つであり、0 または 1 を割り当てる。出力層の活性化関数にはシグモイド関数が用いられ、出力は 0 から 1 の実

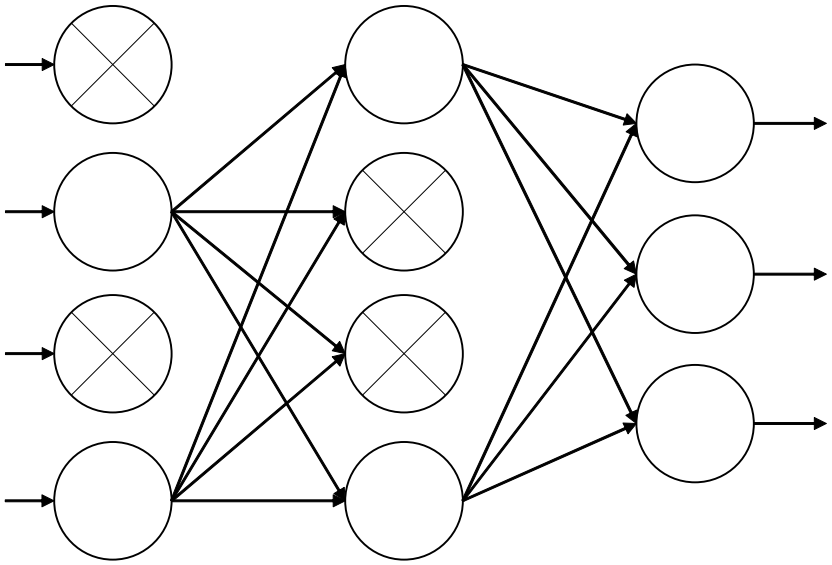

Figure 5.1　Dropout

数となる。この実数は、出力の値に 1 を割り当てた事象である確率を表す。損失関数には交差エントロピー誤差が使用される。

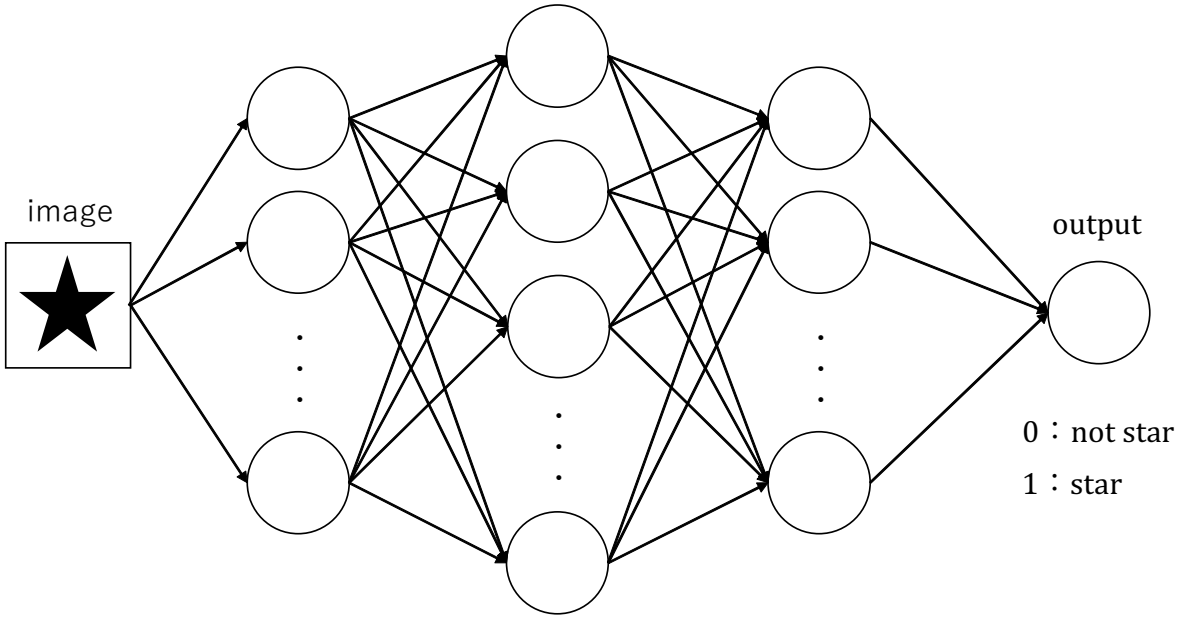

Figure 5.2　Binary-classification problem

5.7.2　多クラス分類問題

多クラス分類問題は、Figure 5.3 のように対象を複数の種類に分類する問題である。分類されたそれぞれの集合はクラスと呼ばれる。ニューラルネットワークで多クラス分類を予測する場合、出力層のユニット数は分類するクラス数となる、各クラスには 0 から

はじまる整数を割り当てる。この整数のことをラベルと呼ぶ。出力層の活性化関数には Softmax 関数が用いられ、誤差関数は交差エントロピー誤差が使われる。訓練データの正解ラベルは、1 つの数値であるためそのまま正解データとしては使うことが出来ない。そこで、正解ラベルに対応するユニットのみ値を 1 とし、それ以外を 0 とする one-hot ベクトル表現が用いられる。

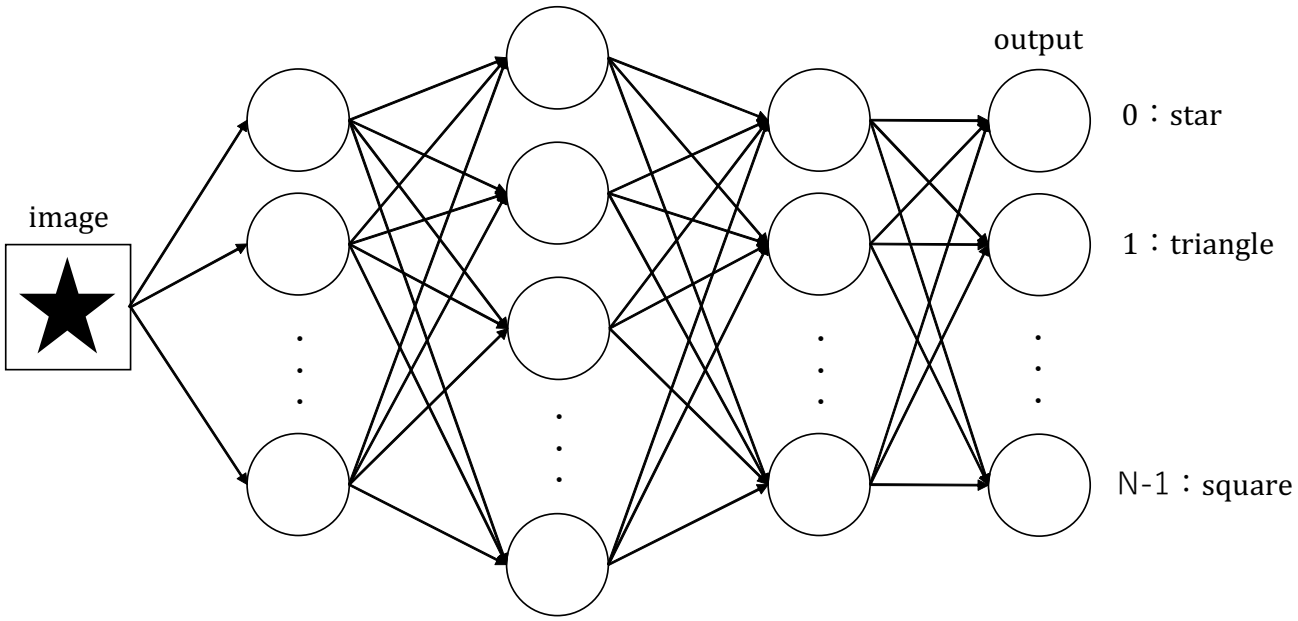

Figure 5.3　Multi-classification problem

5.8　方策ネットワーク

方策ネットワークとは、Google DeepMind によって開発されたコンピュータ囲碁プログラムである AlphaGO Zero で用いられたネットワークの一つである。[8] 教師付き学習フェーズとも呼ばれ、教師データとして与えられた状態 s (ゲームにおける盤面など) を、次に打たれる手 a の確率が最も高くなるような確率分布 $p_\sigma(a|s)$ を出力するように確率的勾配降下法を用いて学習させるものである。[9] 具体的には、畳み込みニューラルネットワークを使い、最終層の出力結果の活性化関数を Softmax 関数にし、確率解釈を行っている。

第6章　MNISTによる分類

6.1　Chainer

Chainerとは、Preferred Networks社が開発したニューラルネットワークを実装するためのPythonライブラリである。これは、CUDAというNVIDIA社GPU向けの汎用並列コンピューティングプラットフォームをサポートしており、GPUを利用した高速な計算が可能である。[10] また、環境構築が他のライブラリと比べ容易であることや柔軟な記法により様々なタイプのニューラルネットを実装可能というメリットがある。

6.2　MNIST

MNISTとは、手書き数字の画像データセットであり、機械学習の分野で最も有名なデータセットの1つである。MNISTデータセットは、Figure 6.1に示すように、0から9までの数字画像から構成される。訓練データ画像が60,000枚、テストデータ画像が10,000枚用意されており、それらの学習と推測を行う。28 × 28のグレー画像で、各ピクセルは0〜255の値をとる。[11] ここで言う訓練データとは、実際に学習を行うデータであり、テストデータとは、正しく分類がされた比較用のデータのことを指す。

Figure 6.1　MNIST

6.3　MNISTを用いた畳み込みニューラルネットワーク

Chainerを用いて畳み込みニューラルネットワークを構築した。ネットワークは入力層、畳み込み層及びプーリング層がそれぞれ2層、全結合層からなる。なお、6章及び7章ではChainer4.3.1・CUDA9.0・Python3.5.5の環境でプログラムの実行を行う。

6.3.1 ライブラリのインポート

```
import numpy as np
import chainer
from chainer import cuda, Function, gradient_check, report, training,
utils, Variable
from chainer import datasets, iterators, optimizers, serializers
from chainer import Link, Chain, ChainList
import chainer.functions as F
import chainer.links as L
from chainer.datasets import tuple_dataset
from chainer import training
from chainer.training import extensions
```

Chainer でニューラルネットワークの定義に必要なパッケージをインポートした。プログラムの簡略化のために、chainer.functions 及び chainer.links をそれぞれ F、L としてインポートを行った。

6.3.2 畳み込み層の定義

```
self.cn1 = L.Convolution2D(1, 20, 5)
self.cn2 = L.Convolution2D(20, 50, 5)
```

Chainer で畳み込み層は上記のように定義される。畳み込み演算は、画像処理で言うところの「フィルタ演算」に相当する。畳み込み演算の例を Figure 6.2 に示す。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とと同じである。入力データは縦・横形状を持つデータで、フィルタも同様に、縦・横方向の次元を持つ。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。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。ここで言うウィンドウとは、灰色の部分を指す。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。なお、引数は左から順に入力チャンネル (次元) 数、出力チャンネル数、フィルタサイズとなる。

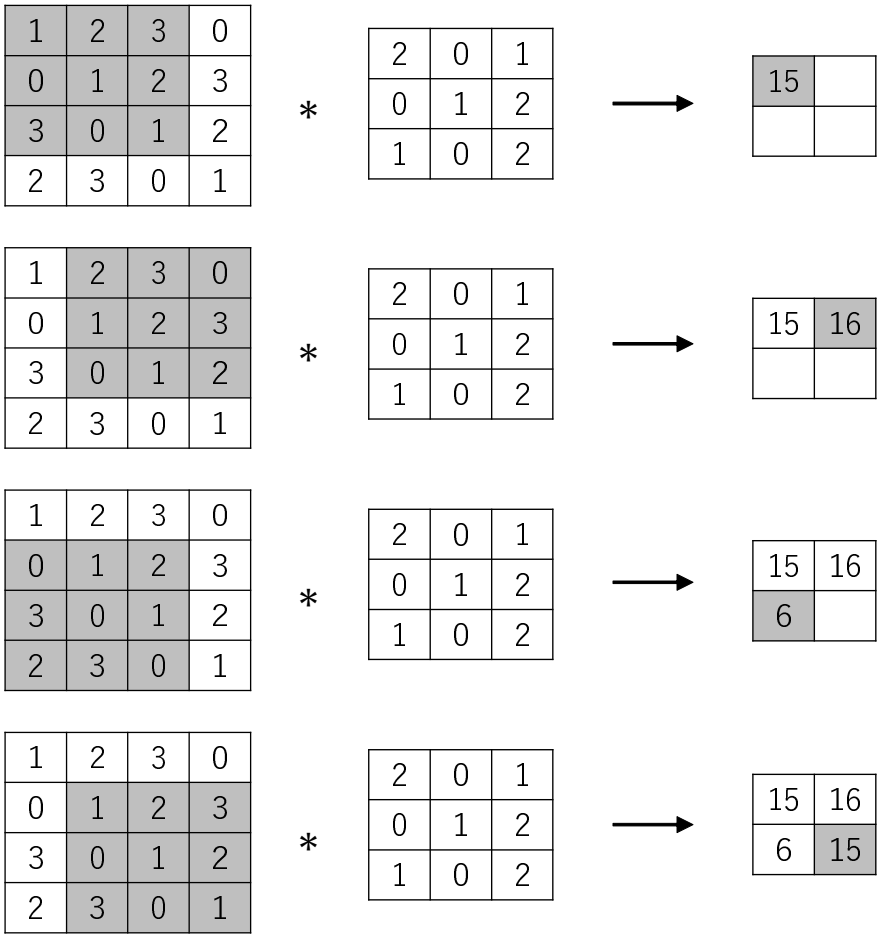

Figure 6.2　Procedure of calculating convolution operation

6.3.3　プーリング層の実装

```
h1 = F.max_pooling_2d(F.relu(self.cn1(x)), 2)
h2 = F.max_pooling_2d(F.relu(self.cn2(h1)), 2)
```

上記のようにプーリング層を実装した。ここでは、畳み込み演算を行い、ReLU 関数に入力を行ったデータに対し Maxpooling 法を用いた。プーリングは、縦・横方向の空間を小さくする演算である。Maxpooling 法とは Figure 6.3 に示すように、例えば、2 × 2 の領域の中での最大値を取り出す処理を行い、空間サイズを小さくしている。なお、引数は左から順番にプーリングを行う対象、領域の大きさである。

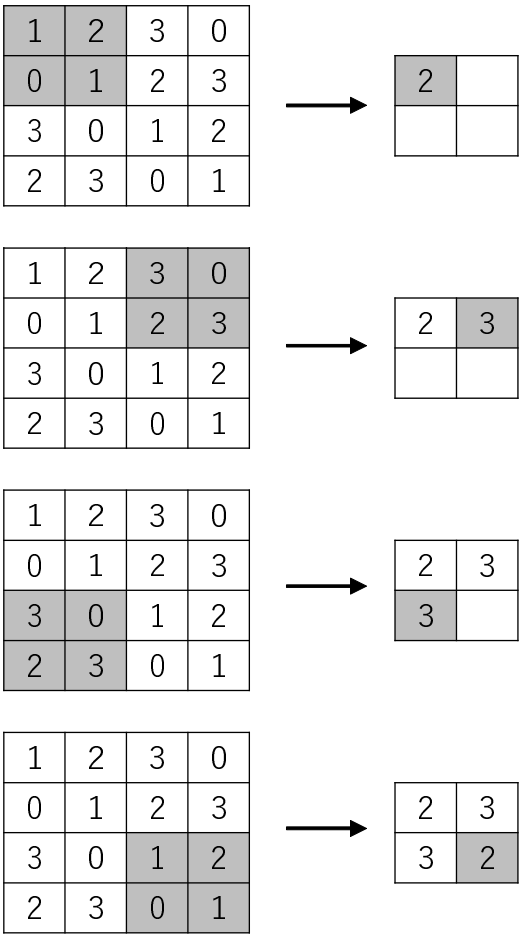

Figure 6.3　Maxpooling procedure

6.3.4　GPU による処理

```
chainer.cuda.get_device_from_id(0)
model.to_gpu()
```

定義したネットワークのモデルを作成し、モデルを GPU に転送した。モデルのパラメータの初期値は自動的に調整されるため、初期化に関するコーディングは行っていない。

6.3.5　最適化手法の設定

```
optimizer = chainer.optimizers.Adam()
optimizer.setup(model)
```

勾配降下法に使用する最適化手法を設定した。ここでは Adam 法を用いた。

6.3.6　学習結果

このような畳み込みニューラルネットワークを構築し 30epoch 学習させた際の正答率及び損失をそれぞれ Figure 6.4 に示す。この条件での学習の正答率は 0.9906 程度になることが示された。

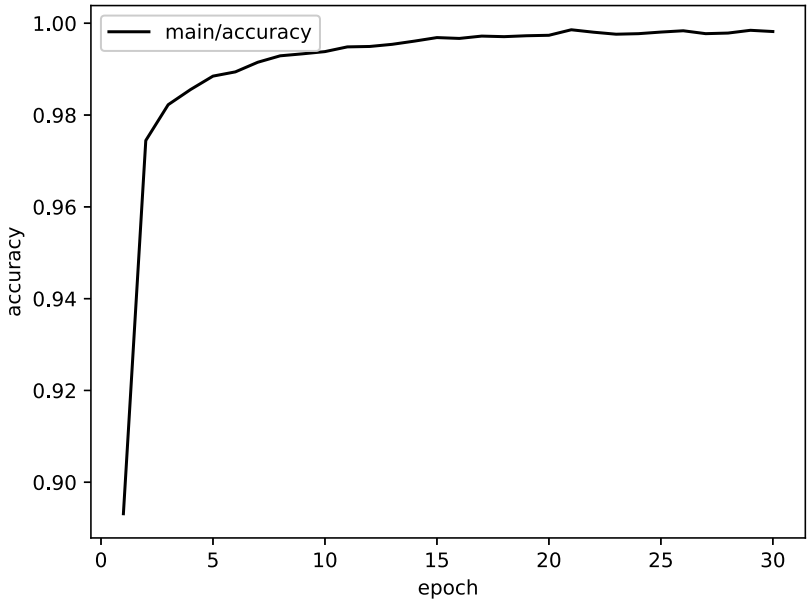

Figure 6.4　Accuracy of MNIST by convolution neural network

第7章　実験

7.1　実験準備

7.1.1　floodgate

floodgate とは 2008 年に公開された shogi-server 上で将棋プログラムを連続対戦させる企画である。プログラムの強さを示す指標として rating が存在する。

7.1.2　訓練データの準備

この実験で使用する畳み込みニューラルネットワークは教師あり学習であるため、教師データが必要不可欠である。棋譜データについてはコンピュータ将棋の対局サイトである floodgate の棋譜を使用する。これをコンピュータ将棋対局所 [12] からダウンロードし、使用する。なお、棋譜は 2016 年度のものを使用する。これには 80141 個の棋譜データが含まれている。floodgate の棋譜には、教師データとして適切でない棋譜も含まれているため、投了で対局終了、手数が 50 手以上、対局プログラムの rating が 2500 以上という条件をすべて満たす棋譜を抽出した。また、学習の精度を評価するための棋譜を無作為に抽出し、教師データとは別に用意した。その結果、教師データは 26782 局、テストデータは 2976 局となった。

7.1.3　データの入力

局面は盤上の駒と持ち駒から構成されている。盤上の駒は、Figure 7.1 に示す通り、駒の種類ごとにチャンネルを分けて、各チャンネルは駒の座標を表す 9 × 9 の 2 値画像とした。また、成り駒は別の駒として扱い、先手と後手で別々のチャンネルに分けた。また、持ち駒は、種類ごとに最大枚数分のチャンネルを割り当てた。Figure 7.2 に示す通り、持ち駒がある場合は、すべて 1 の画像とし、持ち駒がない場合はすべて 0 の画像とした。また、先手と後手で別々のチャンネルに分けた。

7.1.4　出力

出力は指し手の座標に 1 つのラベルを割り当て、多クラス分類問題として扱う。ただし、将棋の場合、駒の移動があるため、移動元の座標の考慮が必要である。これは移動先の座標のみでは、移動元の駒が一意に決まらないことがあるからである。

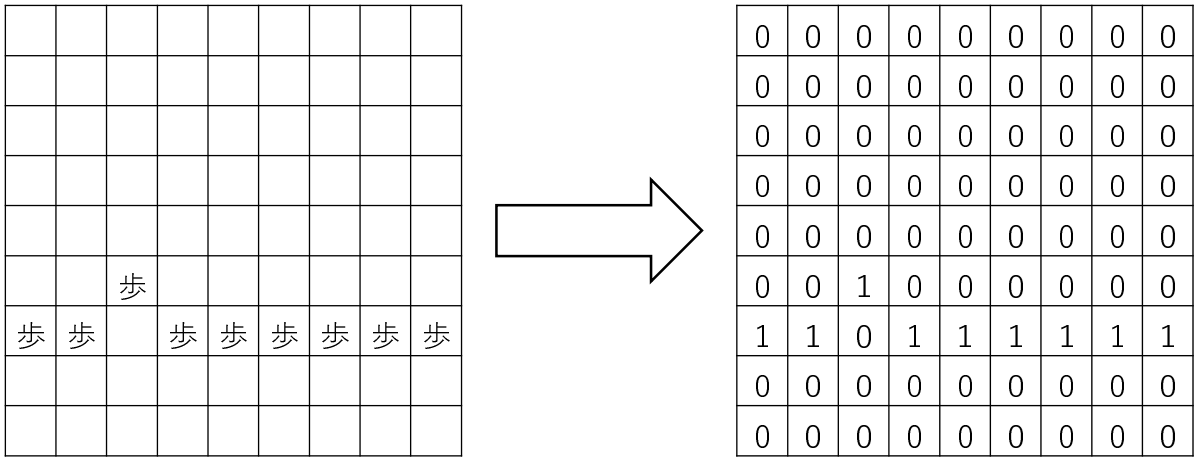

Figure 7.1　Example of the first “Fu” channel

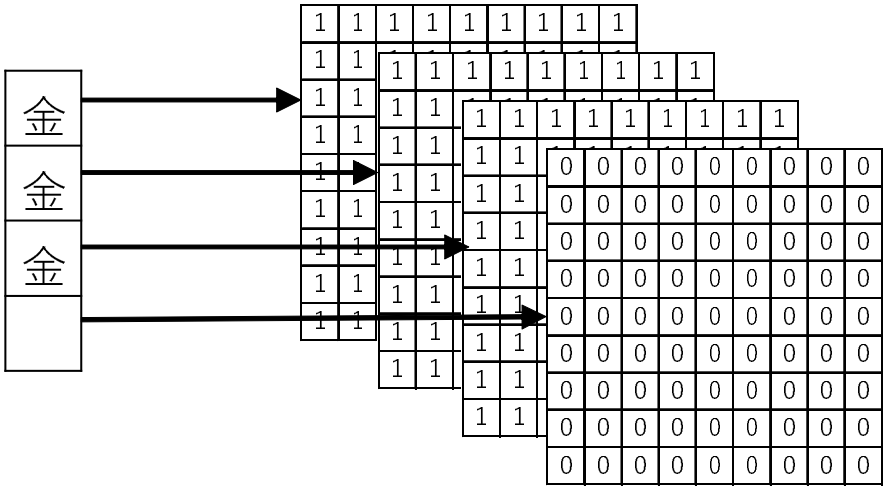

Figure 7.2　Example of the channel of “Kin” in hands

7.1.5　移動元の考慮

移動元をそのまま座標として扱うと、移動元と移動先の座標の組み合わせで 6561 通りのラベルが必要となるが、多くのラベルを扱うと GPU の必要メモリ、計算速度、学習効率、予測精度に悪影響があると考えられる。そこで、移動元を方向のみで扱うようにする。将棋では、桂馬を除く駒には飛び越しがないため、駒の種類と移動方向がわかれば移動元を一意に決めることができる。移動方向は、移動先の駒を中心とした 8 方向と桂馬の動きを加えた 10 方向とする。持ち駒には移動元がないため、駒の種類ごとに 7 種類のラベルを割り当てる。また、駒が成る場合には、別の移動として 10 種類のラベルを割り当てる。移動方向と移動先の座標を組み合わせると、出力ラベル数は 2187 通りとなる。約 $\frac{1}{3}$ に減らすことで十分な精度で予測することが可能であると考えられる。

7.1.6　環境設定

学習における環境設定を以下に示す。

- CPU:Intel Core i7 6700
- GPU:Nvidia Geforce GTX 1070 6GB
- OS:Windows7 Professional
- ニューラルネットワークの作成には、Python3.5.5 および Chainer4.3.1 を用いる。

7.2　CNN に関する研究

ニューラルネットワークの構成は AlphaGo の論文[8)] の構成を参考にして 13 層の CNN とする。通常、CNN で画像認識を行う場合は、プーリング層を使用するが、今回は使用しない。これは、プーリング層の目的が位置のずれを曖昧にするためであるため、今回のような将棋の指し手予測には不向きであるからである。構成したニューラルネットワークを Table 7.1 に示す。入力層と中間層の畳み込み層のフィルターサイズは 3 × 3 とする。また、中間層で画像サイズが変わらないように幅 1 のパディングを行う。中間層の活性化関数は ReLU 関数とする。出力層では、1 × 1 のフィルター 27 枚に、座標ごとに異なるバイアスを加え、Softmax 関数でラベルごとの確率を出力する。学習する epoch 数は 1 とする。

Table 7.1　Structure of 13layer CNN

number of layers	13
number of filters	192
stride	1
optimization method	SGD
pooling layer	unused
activation function	ReLU

学習モデルにテストデータによって評価した結果を Figure 7.3 に示す。Table 7.1 の構成における平均正答率は 0.344、平均損失 3.181 はであった。ここで損失とは、教師データと出力にどれだけの差があるかの尺度で、これを最小にするように学習を進める。この正答率は将棋の指し手での正答率であると考えると、それほど低いものではないとい

える。[8)]

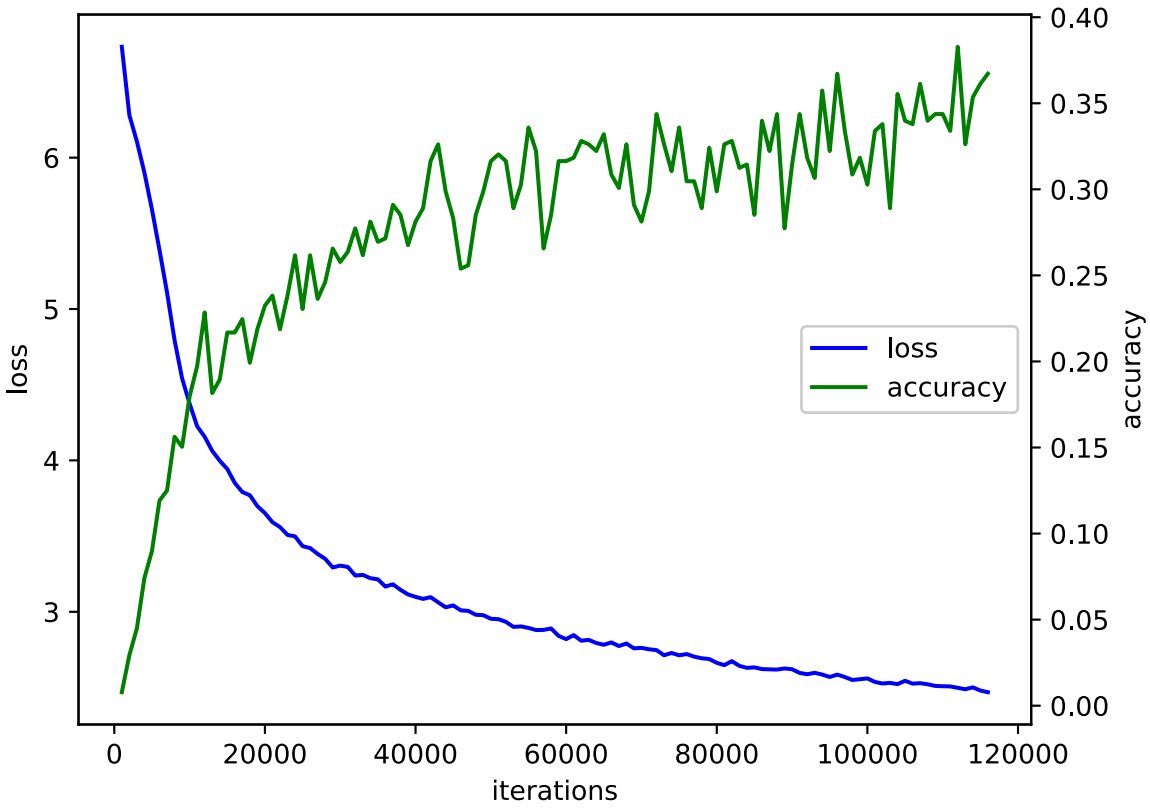

Figure 7.3　Accuracy and error during training

7.2.1　CNN の層数についての研究

次に、13 層のニューラルネットワークの畳み込み層の数を変化させた時の平均正答率、平均損失、学習にかかった時間を Table 7.2 に示す。

層が少ないと学習が足りず、正答率が低くなり損失も大きくなった。その後は層を増やすと正答率が上がり、損失が少なくなったが、8 層よりも多いモデルでは層が増えるほど正答率が下がり、損失も大きくなった。よって、層を増やすだけでは、ニューラルネットワークの学習精度を上昇させることは困難である。また、入力データは同一であるが、層を増やすと正答率が低くなったことから、過学習による精度の低下ではなく非効率な学習が行われていたと考えられる。学習にかかった時間は、層が増えるにつれて長くなった。これは更新する重みの数が増えていく為であるからと考えられる。

Table 7.2　Result of layer experiment

number of layers	accuracy	loss	training time
3	0.301	3.241	42’14”
5	0.350	2.849	47’42”
8	0.364	2.839	54’27”
10	0.356	2.840	56’28”
13	0.346	3.162	1:10’25”
18	0.316	3.383	1:25’19”
23	0.320	3.564	1:43’21”
28	0.310	3.761	2:03’30”

7.2.2　CNN のフィルタ枚数についての研究

次に、8 層のニューラルネットワークのフィルタの数を変化させた時の平均正答率、平均損失、学習にかかった時間を Table 7.3 に示す。

Table 7.3　Result of filter experiment

number of filters	accuracy	loss	training time
64	0.328	3.071	55’05”
128	0.357	2.907	55’00”
192	0.364	2.839	54’27”
256	0.365	2.798	55’41”
512	0.378	2.730	1:31’25”

フィルター枚数が多いほうがより高い正答率を得られた。学習時間は、フィルター枚数が 64〜256 枚ではほとんど変わらず、512 枚で急激に増えた。これはフィルター枚数が 256 枚までは GPU による並列処理が可能であったが、512 枚では GPU で並列不可能な処理量になったためだと考えられる。GPU の並列処理性能はハードウェアの性能によって決まるため、精度のみではなく学習時間とのバランスを考慮して、パラメータを決める必要がある。

7.3 Residual Network による多層化の研究

これまでの実験より、CNN を深層化した場合、精度が下がるという課題があった。そこで残差を用いて深層化を可能にする Residual Network(ResNet) を用いて実験を行なった。ResNet は直接最適な写像になるように学習するのではなくショートカット接続によって残差の写像が最適になるように学習を行うニューラルネットワークである。ショートカット接続はいくつかの層をスキップする恒等写像で、パラメータの追加がなく、計算も複雑にならず、パックプロパゲーションも可能であるため、実装が簡単である。構築した ResNet は Figure 7.4 のブロックを繰りかえす構成となっている。畳み込み層の数は 1 ブロックにつき 2 層である。ショートカット経路においても誤差は逆伝播する。フィルター枚数は 192 枚とし、第 7.2.1 項で構成した層の数に近くしたときの平均正答率、平均損失、学習にかかった時間を Table 7.4 に示す。なお、括弧内の数字は入力層と出力層を含めたニューラルネットワークの層の総数である。

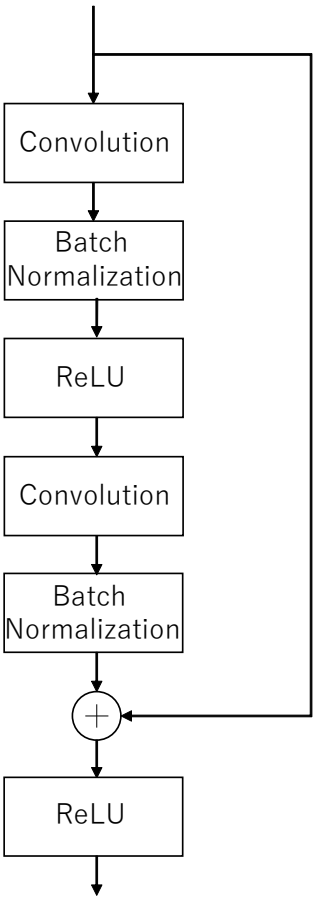

Figure 7.4　Structure of ResNet

Table 7.4　Result of ResNet experiment

number of blocks	accuracy	loss	training time
1 (4)	0.231	4.592	57’46”
3 (8)	0.369	3.107	1:11’30”
4 (10)	0.379	3.063	1:30’39”
5 (12)	0.380	3.046	1:31’38”
8 (18)	0.370	3.043	2:06’40”
10 (22)	0.374	3.067	2:27’42”
13 (28)	0.368	3.039	3:13’29”

通常の CNN の結果に比べて、0.02 程度の精度の向上が現れ、層が多いニューラルネットワークほど精度の向上が見られた。これは、ショートカット接続により非効率な演算を行うことがなくなったからであると考えられる。しかし、学習にかかる時間は、1.3 倍程度になった。これは残差の写像が最適になるようにする演算を行なっているからだと考えられる。

第8章　結論

本研究では、将棋の既存の棋譜データから、各手数での盤面評価を行う畳み込みニューラルネットワークを Chainer を用いて構築した。また、畳み込みニューラルネットワークの深層化による精度の向上を試みた。結果として、畳み込みニューラルネットワークを深層化した場合、ある程度の層数までは精度が向上するが、それ以降は学習が非効率的になり精度が低下する結果が得られた。その問題を解決するために、残差を利用した Residual Network を実装し、深層化による精度の向上を実現できた。

今回の実験で用いたニューラルネットワークは、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、同様な手法で学習可能な画像や音声認識、チェスや囲碁などの様々な分野において効果的であると予想される。ただし、ニューラルネットワークの構成には工夫が必要である。フィルター枚数を増やすと精度が向上したが、0.01 程度の向上に対し、学習にかかる時間は約 1.7 倍長くなった。囲碁などのより盤が大きいものではこの時間がさらに長くなることが予想されるため、これらのバランスを考えニューラルネットワークを構築する必要がある。

今回の実験では、ニューラルネットワークの構築に Chainer ライブラリを用い、GPU での処理のために CUDA を用いたが、Chainer では直感的な計算グラフの構築が可能であり、GPU を利用した非常に高速な演算ができるため、新規のネットワークを構築したい場合、非常に有用である。

Chainer をはじめとして、TensorFlow や Keras など、今日では数多くの機械学習向けライブラリが存在するが、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、ビジネス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。

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小寺智仁氏、長澤紘汰氏、長尾彪真氏に厚く御礼を申し上げます。

参考文献

1) 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).
2) 山下 隆義 :イラストで学ぶディープラーニング, 講談社 (2016).
3) Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun : Deep Residual Learning for Image Recognition, arXiv.org cs arXiv:1512.03385 (2015).
4) 涌井 良幸, 涌井 貞美 :ディープラーニングがわかる数学入門, 技術評論社 (2017).
5) L. Bottou, O.Bousquet, "The Tradeoffs of Large-Scale Learning", Optimization for Machine Learning (2011).
6) 舩橋 聡太 :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 (2018).
7) Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
8) David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel and Demis Hassabis : Mastering the game of Go without human knowledge, Nature 550, pages 354–359 (2017).
9) 山岡 忠夫 :将棋 AI で学ぶディープラーニング, マイナビ出版 (2018).
10) Chainer, https://chainer.org/, 2018 年 10 月 30 日アクセス.
11) MNIST, http://yann.lecun.com/exdb/mnist/, 2018 年 10 月 30 日アクセス.
12) コンピュータ将棋対局所, http://wdoor.c.u-tokyo.ac.jp/shogi/ 2018 年 10 月 30 日アクセス.