ディープラーニングの株価予測への応用

池田　欽一　・　林田　実

概要

本研究では、近年飛躍的な進化を遂げたニューラルネット技術（ディープラーニング）の株式投資へ応用可能性というテーマで分析を行っている。ディープラーニングの学習手法やシミュレーション分析を行う際の実装方法、実際の株価を用いたシミュレーション分析の方法と結果についてまとめている。

1　はじめに

本研究では、ディープラーニングを用いた株価の予測の可能性について分析を実施する。ディープラーニングは、簡単に言えば従来より研究されてきた神経細胞を数理モデル化したユニットによるネットワーク、いわゆるニューラルネットワークの層を増やし、表現、学習能力を高めたものである。単に層を増やすだけではユニット間のつながりの強さを表すウェイトの修正（学習）において、学習手法の1種であるバックプロパゲーションのみでは修正項の急激な減少などにより適切な学習が不可能であったが、ある種の事前学習の方法を採用することにより、多層ネットワーク学習でも効率的な学習が可能となることが示された。音声認識、コンピュータビジョン、翻訳や囲碁や将棋などの推論手法などへの応用が研究されているなど、人工知能技術の大幅な進展に貢献している。

ディープラーニングは大量のデータ、いわゆるビッグデータを用いた学習をすることにより、学習に用いていないデータに対しても適切な予測能力を示すことができ、これまでコンピュータが不得意であった、経験則に基づかない予測、判断が可能となってくる。本研究では、この予測能力が、予測が困難な問題と考えられる株式予測へ有効であるのかを検証することとする。

株価は正規分布やベキ分布に従うランダムな現象としてモデル化されることが多いが、ディープラーニングは非線形性を持った多数のユニットをネットワーク状につないだものであるので、複雑な非線形システムの学習、予測に強く、従来はランダムな現象と考えられている対象を、決定論的な非線形現象として分析が可能である。株価以外でも非線形性が強いと考えられる経済データへの適用が有効であることが考えられる。

以下、2. ではディープラーニング手法、特に本研究で用いる方法について述べ、3. ではディープラーニングを株価予測へ応用するための実装方法や用いる株価データ、パラメータ設定について述べる。4. ではシミュレーション結果とその考察についてまとめ、5. でまとめとする。

2　ディープラーニングとは

ディープラーニング（深層学習）とは、神経細胞（ニューロン）をモデル化したものを入力から出力までのいくつかの層に分けて接続し、入力層に入力されたデータがネットワーク内で非線形変換され、出力層からネットワーク全体の出力がなされるフィードフォワードニューラルネットワーク

の層を増やしたシステムである。ニューラルネットワークでの接続強度（ウェイト）の修正（学習）方法としてバックプロパゲーション（誤差逆伝搬法）が用いられることが多いが、層を増やすとより入力層側のウェイトの修正量が小さくなり、適切に学習が進まない問題が指摘されてきた。ディープラーニングでは事前学習によりデータの特徴を事前に学習し、その特徴を抽出するためのウェイトを、バックプロパゲーションの初期値とすることで効果的な学習ができることが特徴である [1]。

2.1 ニューラルネットワーク

本研究では前節で述べた様に、ニューラルネットワークの中でもニューロンを層ごとにならべるフィードフォワードネットワークを用いる [2, 3, 4]。フィードフォワードネットワークでは入力層に入力データを与え、出力層に望ましい値（教師データ）が出力されるよう学習を進める。入力層と出力層の間の各層は中間層と呼ばれ、中間層を増やすほどデータを学習する能力、データの表現力は増加していく。入力層のニューロン数は入力データの次元と同じ数とし、出力層は教師データの次元と同じ数とし、中間層は任意の数とすることができる。フィードフォワードネットワークでは、中間層と出力層は（より入力層側の）1 つ隣の層の各ニューロンからのみ受け取り、1 つ前の層のニューロンの出力値に接続強度であるウェイトをかけたものを入力値とする。出力層の出力と教師データの誤差が小さくなるようにウェイトを修正していくことでネットワーク全体の学習が進められる。

2.1.1 ニューロン

ニューロンとは神経細胞を次のようにモデル化したものである。

$$u_j = \sum_{i=1}^{I} w_{ji} x_i + b_j, \tag{1}$$

$$z_j = f(u_j). \tag{2}$$

ここで、u_j は第 j ニューロンの内部状態を表し、$x_i (i = 1, 2, \ldots, I)$ であらわされるニューロンへの入力（1 つ前の層のニューロンの出力値）、w_{ji} は 1 つ前の層の第 i ニューロンから第 j ニューロンへの接続の強さを表すウェイト、b_j は閾値から計算される。z_j はニューロンの出力値で、内部状態に出力関数（活性化関数）$f()$ を適用した値で求められる。ただし、入力層の第 k ニューロンは、次のように入力されたデータ（学習データ）をそのまま出力することとする。

$$\mathrm{z}_k = x_k. \tag{3}$$

$x_k (k = 1, 2, \ldots, K)$ はニューロンの入出力値であり、学習データでもある。つまり、K は学習データの次元を表している。

活性化関数には次のようなシグモイド関数や双曲線正接関数 $(tanh())$ がよく用いられる。シグモイド関数は、次のような式であらわされる。

$$f(u) = \frac{1}{1 + e^{-u}}. \tag{4}$$

図 1 にはニューロンモデルの模式図を示している。

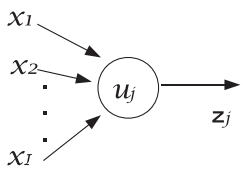

図 1: ニューロンのモデル

2.1.2 フィードフォワードニューラルネットワーク

上記のようなニューロンをまとめた層をシステムの入力側から出力へ、入力層、任意の数の中間層、出力層と並べ、入力層へは学習データをそのまま与え、中間層と出力層は 1 つ入力側の層の全ニューロンからのみ入力を受け取るネットワークをフィードフォワードニューラルネットワークと呼ぶ。第 l 層の内部状態 $\boldsymbol{u}^l = (u_1^l, u_2^l, \ldots, u_j^l, \ldots, u_{J^l}^l)'$ は次のように表される。

$$\boldsymbol{u}^l = W^l \boldsymbol{z}^{l-1} + \boldsymbol{b}^l. \tag{5}$$

ここで、$\boldsymbol{z^{l-1}} = (z_1^{l-1}, z_2^{l-1}, \ldots, z_i^{l-1}, \ldots, z_{I^l}^{l-1})'$ は 1 つ入力層側の第 $l-1$ 層のニューロンの出力からなる第 l 層への入力ベクトルで、$\boldsymbol{b^l}$ は第 l 層の各ニューロンの閾値からなる列ベクトルである。W は次のような第 $l-1$ 層と l 層のニューロン間のウェイト行列である。

$$W^l = \begin{pmatrix} w_{11}^l & w_{12}^l & \cdots & w_{1I^l}^l \\ w_{21}^l & w_{22}^l & \cdots & w_{2I^l}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{J^l1}^l & w_{J^l2}^l & \cdots & w_{J^lI^l}^l \end{pmatrix}. \tag{6}$$

第 l 層の出力 $\boldsymbol{z^l} = (z_1^l, z_2^l, \ldots, z_j^l, \ldots, z_{J^l}^l)'$ は次のように表される。

$$\boldsymbol{z}^l = \boldsymbol{f}(\boldsymbol{u}^l). \tag{7}$$

ここで、$\boldsymbol{f}(\boldsymbol{u})$ は活性化関数ベクトルで次のようなものである。

$$\boldsymbol{f}(\boldsymbol{u}^l) = (f(u_1^l), f(u_2^l), \ldots, f(u_{J^l}^l))'. \tag{8}$$

図 2 にはフォードフォワードニューラルネットワークの例を示している。入力層にデータが与えられ、この図で見て左から右へ情報が伝わり出力層の出力がシステム全体の出力となる。教師データが連続値である場合は出力層のユニットは 1 つとして、学習後のこの 1 ユニットの出力を予測値として扱うことが一般的である。カテゴリデータのクラス分類の場合、出力層のユニット数をグループ（クラス）の数と同一とし、各ユニットを各グループに割り当て、学習後に入力データを与え、最も出力の大きいユニットに割り当てられたグループに所属すると予測することが一般的である。

2.1.3 バックプロパゲーション

フィードフォワードニューラルネットワークの学習方法については、Rumelhart により、ある入力データを与えた場合のネットワークの出力値と入力したデータに対する望ましい出力（教師データ）

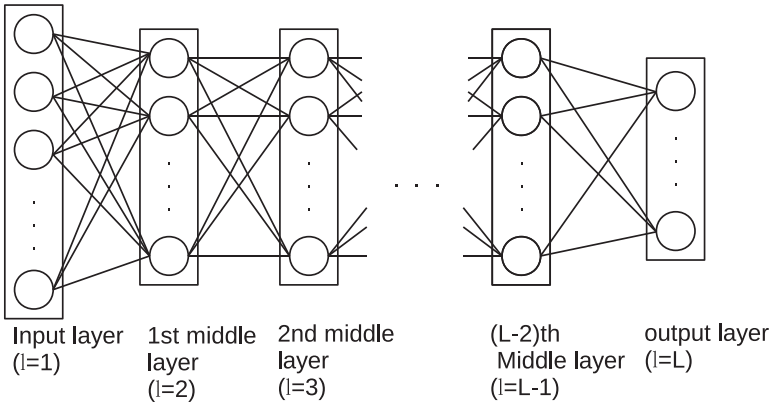

図 2: フィードフォワードニューラルネットワーク

の誤差を出力層側から入力層側へ伝搬しながらウェイトを修正していく誤差逆伝搬法（バックプロパゲーション法）が示された [4]。ある入力データとそれに対する望ましい出力の組みを学習データと呼ぶことにする。

第 p 番目の学習データについて、入力データから計算されたネットワークの出力層の出力 $\boldsymbol{z}_p^L$ を並べたベクトルを $o_p = (o_{p1}, o_{p2}, \ldots, o_{pJ})'$ と表し、この出力値と教師データベクトル $\boldsymbol{t}_p = (t_{p1}, t_{p2}, \ldots, t_{pJ})'$ の誤差 E_p を次のように定義する。

$$E_p = \frac{1}{2}\sum_{j=1}^{J}(t_{pj} - o_{pj})^2. \tag{9}$$

この式で 1/2 は偏微分した際に係数が 1 になるようにするためである。

すべての学習データについての誤差は次のように計算される。

$$E = \sum_{p} E_p. \tag{10}$$

この誤差を出力層とその 1 つ入力側の中間層の間のウェイト W^L により偏微分し、最急降下法によりウェイトを修正することによりネットワーク全体は入力データに対応した教師データに近い値を出力するように学習が進んでいく。

ウェイト修正は偏微分を用い実施されるが、すべての学習データについての合計誤差 E は 1 つずつの学習データの誤差の単純な和で表されるので、以下では 1 つずつの学習データによる修正について説明する。また出力関数は、式 (4) のシグモイド関数を仮定する。

出力層（第 L 層）の第 j ニューロンへの第 $L-1$ 層の第 i ニューロンの接続の強さは w_{ji}^L であるが、出力層の誤差 E_p が最も小さくなるように最急降下法を用いて w_{ji}^L を修正していく。E_p を $w_j^L i$ により偏微分すると、

$$\frac{\partial E_p}{\partial w_{ji}^L} = \frac{\partial E_p}{\partial o_{pj}}\frac{\partial o_{pj}}{\partial u_{pj}^L}\frac{\partial u_{pj}^L}{\partial w_{ji}^L}. \tag{11}$$

この式の右辺は、出力の変化により誤差がどれほど変化するか (A と表す) とシグモイド関数の偏微分値 (B と表す)、ウェイトの変化によりユニットの内部状態がどれほど変化するかの値 (C と表す)

の積となっている。式 (9) の誤差を用いると、A は次のように表され、これを $-\delta_{pj}^{L}$ と置いておく。

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) = -\delta_{pj}^{L}. \tag{12}$$

B は、次のように表される [4]。

$$\frac{\partial o_{pj}}{\partial u_{ij}^{L}} = o_{pj}(1 - o_{pj}). \tag{13}$$

また、出力層のユニットの出力は次のように表されるので、

$$o_{pj} = f(\sum_{i=1}^{I} w_{ji}^{L} z_i^{l-1} + b_j^{L}). \tag{14}$$

C は、次のようになる。

$$\frac{\partial o_{pj}}{\partial w_{ji}^{L}} = z_i^{l-1}. \tag{15}$$

これらを、式 (11) に代入すると、次のようになる。

$$\frac{\partial E_p}{\partial w_{ji}^{L}} = -\delta_{pj} o_{pj}(1 - o_{pj}) z_i^{l-1}. \tag{16}$$

全サンプルでの誤差 E は式 (16) の和により求められる。

$$\frac{\partial E}{\partial w_{ji}^{L}} = -\sum_{p=1}^{P} \delta_{pj}^{L} o_{pj}(1 - o_{pj}) z_i^{L-1}. \tag{17}$$

この値を用い最急降下法を適用すると、ウェイト（修正前を $w_{ji}^{L(t)}$、修正後を $w_{ji}^{L(t+1)}$ と表す。）は次のように修正される。

$$w_{ji}^{L(t+1)} = w_{ji}^{L(t)} - a\sum_{p=1}^{P} \delta_{pj}^{L} o_{pj}(1 - o_{pj}) z_i^{L-1}. \tag{18}$$

a は学習定数と言われるもので、修正の強度を表している。a が大きくなると 1 回あたりの修正量は大きくなるが、誤差超平面の（グローバル）極小値へと収束せず学習に失敗することがあり、小さくすると 1 回の修正量が小さくなり学習の進度が遅くなり、また（ローカル）極小解から抜け出しにくくなるなどの問題がある。学習定数を自動的に調整する方法も提案されているが、現在でも試行錯誤的に設定することが一般的である。

このウェイト修正式では、全学習データについての誤差の合計値 E を用いて修正しているが、個々の学習データ、あるいは学習データの一部を順に与え学習を進める手法は確率的最急降下法と呼ばれる。この確率的最急降下法は収束までに時間がかかることがあるが、極小解へとらわれる確率を低減することができる。

出力層のユニットの閾値 b_j^{L} についても同様に偏微分を用い最急降下法による修正値（修正前を $b_j^{L(t)}$、修正後を $b_j^{L(t+1)}$ と表す。）を求めると、次のようになる。

$$b_j^{L(t+1)} = b_j^{L(t)} - a\sum_{p=1}^{P} \delta_{pj}^{L} o_{pj}(1 - o_{pj}). \tag{19}$$

上記の $L-1$ の中間層と出力層の間のウェイト修正方法を、$L-1$ 層と $L-2$ 層、さらに入力側のウェイトへと適用し、入力層と 1 つ目の中間層のウェイトまで学習を進めていく。$L-1$ 層と $L-2$ 層の間のウェイト修正量は次の偏微分に学習定数をかけた値として求められる。

$$\frac{\partial E_p}{\partial w_{ik}^{L-1}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial u_{pj}^{L}} \frac{\partial u_{pj}^{L}}{\partial z_{ik}^{L-1}} \frac{\partial z_{pk}^{L-1}}{\partial u_{pk}^{L-1}} \frac{\partial u_{pk}^{L-1}}{\partial w_{jk}^{L-1}}. \tag{20}$$

この偏微分式を用いたウェイト修正（修正前を $w_{ji}^{L-1(t)}$、修正後を $w_{ji}^{L-1(t+1)}$ と表す。）は同様にそれぞれの偏微分を計算しまとめると次のようになる。

$$w_{ji}^{L-1(t+1)} = w_{ji}^{L-1(t)} - a \sum_{p=1}^{P} \delta_{pi}^{L-1} z_{pj}^{L-1} (1 - z_{pj}^{L-1}) z_j^{L-2}, \tag{21}$$

$$\delta_{pi}^{L-1} = \sum_{j=1}^{J} \delta_{pj}^{L} w_{ji}^{L}. \tag{22}$$

閾値の修正（修正前を $b_i^{L-1(t)}$、修正後を $b_i^{L-1(t+1)}$ と表す。）は次のようになる。

$$b_i^{L-1(t+1)} = b_i^{L-1(t)} - a \sum_{p=1}^{P} \delta_{pi}^{L-1} z_{pk}^{L-1} (1 - z_{pj}^{L-1}). \tag{23}$$

$L-2$ 層と $L-3$ 層のウェイトも同様に修正され、入力層と 1 つ目の中間層の修正まで出力層から入力層までのウェイト修正が実施されることにより、ネットワーク全体が与えられた学習データに対応する教師データに近い出力をするように学習が進んでいく。誤差が出力層側から入力層側へ伝えられることから誤差逆伝搬法と呼ばれている。

誤差逆伝搬法をまとめると次のようなウェイト修正方法となる。

$$\delta_{pj}^{l} = \begin{cases} (t_{pj} - o_{pj}), & \text{if } l = L, \\ \sum_i^I \delta_k^{l+1} w_{ji}^{l+1}, & \text{otherwise.} \end{cases} \tag{24}$$

$$w_{ji}^{l(t+1)} = w_{ji}^{l(t)} - a \sum_{p=1}^{P} \delta_{pj}^{l} z_{pj}^{l} (1 - z_{pj}^{l}) z_i^{l-1}, \tag{25}$$

ただし、δ 計算の際、各層のウェイトが大きいとデルタは発散し、値が小さいとデルタは消失する（急速に 0 になる）こととなる。ニューラルネットワークは中間層を増やすほど複雑な現象を学習できるが、このデルタ消失問題により多層のニューラルネットワークでは入力層側のウェイト学習が有効的に進まないことが指摘されてきた。この問題の解決のための事前学習を取り入れたものがディープラーニングとなる。

2.1.4 Adam

Adam(Adaptive moment estimation) とは、少量のメモリで一次勾配のみにより適用可能な効率的な確率的最適化手法で、パラメータごとに勾配の平均、分散の推定値を利用する手法である。

α を学習定数、$\beta_1, \beta_2 \in [0, 1)$ を勾配の平均、分散推定の際の指数的減衰率、$f(\theta)$ を θ をパラメータとした微分可能な確率的目的関数、θ_0 をパラメータの初期値、m_0, v_0 をそれぞれ勾配の平均、分散のベクトル、t を学習の時間ステップとすると、アルゴリズム 1 の様なものとなる [5]。各パラメータの勾配の平均、分散を推定したものを利用し、勾配が疎になる場合や非定常に強い性質を持っている [6]。

本研究では、学習アルゴリズムとして Adam を採用することとする。

Algorithm 1 Adam のアルゴリズム

Require: α：学習定数。
Require: $\beta_1, \beta_2 \in [0, 1)$：勾配の平均、分散推定の指数的減衰率。
Require: $f(\theta)$：パラメータ θ の確率的目的関数。
Require: θ_0：初期パラメータベクトル。
  $m_0 \leftarrow 0$ （平均ベクトルの初期化。）
  $v_0 \leftarrow 0$（分散ベクトルの初期化。）
  $t \leftarrow 0$（学習の時間ステップの初期化。）
  while θ_t が収束するまで **do**
    $t \leftarrow t + 1$（時間ステップを進める。）
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$（$t$ での目的関数の傾きを求める。）
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$（バイアスのある推定平均の更新。）
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$（バイアスのある推定分散の更新。）
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$（平均のバイアス修正。$\beta_1^t$ は β_1 の t 乗を表す。）
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$（分散のバイアス修正。$\beta_2^t$ は β_2 の t 乗を表す。）
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$（パラメータ更新。$\epsilon$ は発散しないための定数。）
  end while
  return θ_t（得られたパラメータを返す。）

2.2 事前学習

ディープニューラルネットワークのより入力側の学習の効率化のために、各ウェイトを層ごとにあらかじめ学習をしておく方法が事前学習である。本研究では事前学習の手法として積算自己符号化器を用いることとする。

2.2.1 自己符号化器

自己符号化器とは、入力層、中間層 1 つ、出力層からなる 3 層のフィードフォワードニューラルネットワークを考え、入力層に与えた学習データを出力層に再現するように学習を進めていくモデルである。つまり、教師データは入力データと同じものとなる。

入力層のユニット数（=出力層のユニット数）より中間層のユニット数が少ない場合、入力データを次元の低いデータへいったん写像し、再度元の次元へ写像することとなり、中間層に入力データの特徴を抽出することが可能となる。より少ない次元で元の情報を含むデータを抽出することから、多変量解析の主成分分析と同様な考え方となる。

入力層のユニット数（=出力層のユニット数）より中間層のユニット数が多い場合には、入力データをより高次元データへ情報を分散して、再度元のデータを再現するモデルとなる。

特に前者の特徴抽出を多層ニューラルネットワークのすべての層の間のウェイトの事前学習に用いたものが次の積算自己符号化器と呼ばれるものである。

2.2.2 積算自己符号化器

多層ニューラルネットワークのウェイトの初期値に、自己符号化器を繰り返し適用し、すべての層の接続ウェイトへ適用したものを積算自己符号化器と呼ぶ。

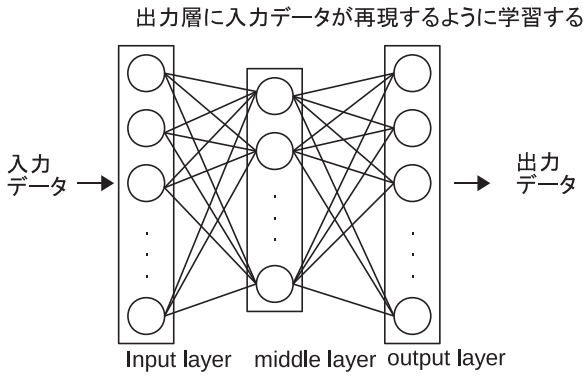

図 3: 自己符号化器

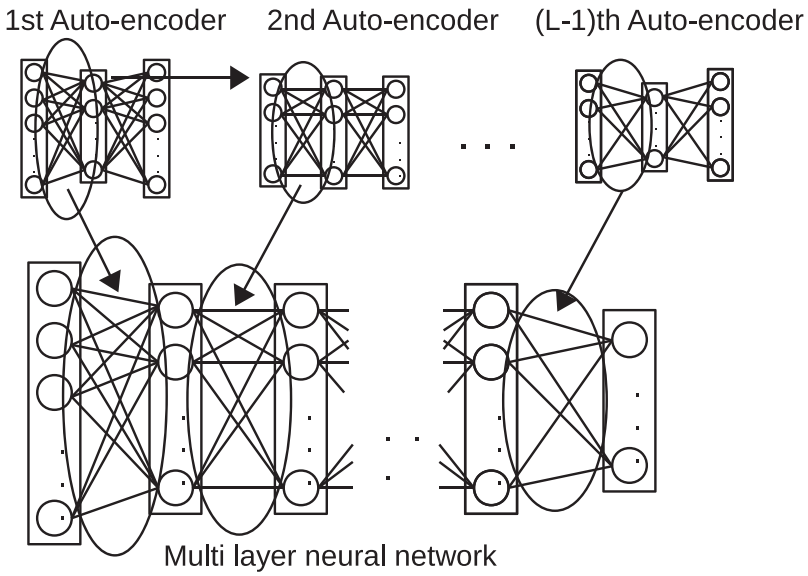

図 4: 積算自己符号化器

図 4 には積算自己符号化器によるフィードフォワードニューラルネットワークのウェイト初期化の例を示している。第 1 の自己符号化器（1st Atuo-encoder）では全体のシステムへの入力データが入力層へ与えられ、出力層でその入力データを再現するように十分な学習をする。学習終了後、自己符号化器の入力層と中間層の間のウェイトを図の下部の全体の多層ニューラルネットワークの入力層と第 1 中間層の間のウェイトの初期値として利用する。これにより、多層ニューラルネットワークへの入力が第 1 中間層で特徴を集約したデータへと変換されることとなる。

同様に学習済みの第 1 の自己符号化器の中間層の各ユニットの出力ベクトルを第 2 の自己符号化器（2nd Atuo-encoder) の入力、および教師データとして用いて十分な学習をする。学習後、第 2 の自己符号化器の入力層と中間層の間のウェイトを多層ニューラルネットワークの第 1 中間層と第 2 中間層の間のウェイトの初期値として利用する。

この初期値決定プロセスを第 $L-1$ 自己符号化器（(L-1)th Atuo-encoder) まで進め、この自己符号化器の入力層と中間層の間の学習済みウェイトを多層ニューラルネットワークの第 $L-1$ 層と出力層の間のウェイトの初期値として利用する。

上記手順により多層ニューラルネットワークのウェイトの初期値が　設定され、その後、入力層に学習データを与え、出力層で教師データを用いたネットワーク全体の学習を進めるとデルタ消失問題が解決され、適切なネットワークウェイトが学習できることが示されている [1]。

3　ディープラーニングの株価予測、学習への応用

株価予測へのディープラーニング応用では、1 分毎の株価情報、および変動率などに加工した情報の現在値と数分前の値（ラグ）を用い、数分後に株価が上がるか、下がる（変わらない含む）かを予測することとする。以下では、用いる株価情報の詳細、および実際のシステム構築方法、パラメータの設定についてまとめている。

3.1　用いる株価データ

ディープラーニングの学習の基本的考え方は大量のデータによる学習を用いることであるので、株価データは年足や月足よりも日足、さらにはティックデータである方がデータ分量という点では適切であると考えられる。ティックデータであるとサンプル発生期間が一定でなく、確定的な期間先の予測という目的には不適合であるので、本研究で用いる株価データは、1 分足データとする。具体的には、個別銘柄の 1 分間の 4 本値（始値、高値、安値、終値）と出来高、高値と安値の差（レンジ）、さらにはこれら値の 1 分間の変化率を変数として用いる。

第 m 変数の変化率 $r_{x_t}^{(m)}$ の計算は、t 期の変数の値を $x_t^{(m)}$ とすると以下のようになる。

$$r_t^{(m)} = \frac{x_t^{(m)} - x_{t-1}^{(m)}}{x_{t-1}^{(m)}}. \tag{26}$$

さらには、日付インデックスとして基準日（2016 年 7 月 28 日）からの経過営業日数、各日ごとの市場が開いてからの経過分数（分数インデックス）も用いる。用いる変数をまとめると以下のようなものとなる。

- $x_t^{(1)}$：基準日からの経過営業日。　例：1,2,3,….
- $x_t^{(2)}$：その日の中で何番目（何分目）のデータか。　　例 1,2,3,…

- $x_t^{(3)}$：始値（1 分足）
- $x_t^{(4)}$：高値（1 分足）
- $x_t^{(5)}$：安値（1 分足）
- $x_t^{(6)}$：終値（1 分足）
- $x_t^{(7)}$：出来高（1 分足）
- $x_t^{(8)}$：レンジ（高値-安値）（1 分足）
- $r_t^{(3)}$：始値変化率（1 分間）
- $r_t^{(4)}$：高値変化率（1 分間）
- $r_t^{(5)}$：安値変化率（1 分間）
- $r_t^{(6)}$：終値変化率（1 分間）
- $r_t^{(7)}$：出来高変化率（1 分間）
- $r_t^{(8)}$：レンジ変化率（1 分間）

また、これら変数のうち $x_t^{(3)} \sim r_{x_t}^{(8)}$ については T_L 期前までのラグ変数も用いることとする。4 期前のラグまで用いる場合のネットワークの入力データ次元（これは入力層のユニット数と同数となる）は、日付、分数インデックス (2) ＋現在値 (12) ＋ 4 期前までのラグ (12 × 4) で 62 次元となる。

1 分足を用いるので、流動性の高い銘柄を用いないと株価 4 本値の各値や出来高の値が取得できないことが多くなるので、1 分間で取引が多数成立するよう流動性の高い銘柄を選ぶことが適切となるが、本研究ではソフトバンクグループ (9984) を用いることとした。教師データはタイムステップ T_{pred} 先に、1 分足終値基準で上がる (1 であらわす) か、または下がる (0 で表し、終値が変化しない場合も含む) のカテゴリデータを用いることとする。用いる株価の期間は、2016 年 7 月 28 日から 2016 年 10 月 11 日までの営業日の株価を学習させ、2016 年 10 月 12 日から 2016 年 10 月 27 日までの株価は学習には用いず、学習したネットワークの汎用的予測能力の検証用データとして用いる。

3.2　実装方法

ディープラーニングを用いた株価予測シミュレーションでは Chainer フレームワーク（ライブラリ）を用いて実装をする [5]。Chainer とは、ネットワークを柔軟に表現できるフレームワークで、ディープラーニングの計算では、計算結果のみでなく計算過程や傾きを保存しておき、ウェイトの修正量計算に利用していることが特徴である。さらに、GPU を用いた並列計算が容易に実施可能で、本研究のシミュレーションでは、CPU のみによる計算のおよそ 10 倍の計算速度となり、並列化は大量のデータ（ビッグデータ）を使い大量の計算を実施するディープラーニングシミュレーションでは必須の条件となる。また、Chainer 以外にもディープラーニングのフレームワークは多数あるが、Chainer は単純なネットワークであっても、複雑なネットワークであっても柔軟に設計ができることや、ネットワーク設定を別途準備するなどの必要もなくプログラミング言語 Python のみで記述が可能で、開発のしやすさも特徴である [5]。

なお、本研究では Python、Chainer、および GPU 並列計算用のライブラリである NVIDIA 社の CUDA ライブラリを Microsoft Windows10 へインストールしたシステム上でシミュレーションを実施した。

3.3 ニューラルネットワークの設計

用いるニューラルネットワークは、2.1.2 で説明したフィードフォワードネットワークであり、入力層のユニット数は入力されるデータの次元と同数となる。本研究では前節で示したようにラグ 4 まで用いる場合には 62 ユニットとなる。中間層の数は試行錯誤的に決められることが多いが本シミュレーションでは 8 層とし、各中間層のユニット数は $m1, m2, \ldots, m8$ と表すこととする。出力層のユニット数は、設定したタイムステップ後に株価が上がるか下がる（変わらない含む）かによる 2 カテゴリの予測とするため 2 ユニットとし、どちらのユニットの出力が大きいかによりカテゴリの予測を行う。つまり、1 つ目のユニット（0 番ユニット）の出力の方がもう一方のユニットの出力より大きければ株価予測は「下がる（変わらない含む）」、2 つ目のユニット（1 番ユニット）の出力の方が大きければ「上がる」という予測となる。また、本シミュレーションでは 3 分後に 1 分足終値が上がるか、下がる（変わらない含む）かを予測することとする（$T_{pred} = 3$）。

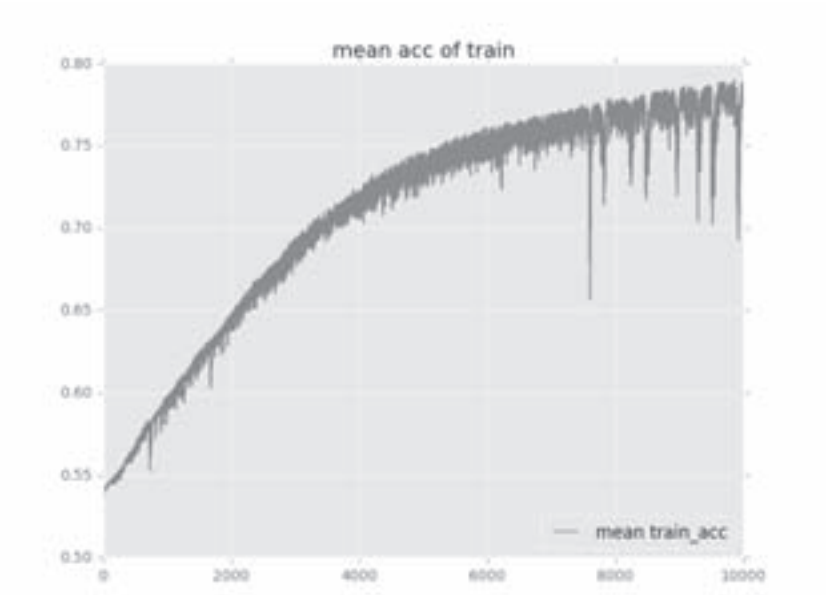

図 5: 学習データの正解率推移

中間層の数は 8 層で固定とするが、各層のユニット数は $m_1 = m_2 = m_3 = m_4 = m_5 = m_6 = m_7 = m_8 = 37$ とすべて 37 とした。この中間層ユニット数も試行錯誤的に決められるが用いるデータの性質により設定することが重要である。層の数やユニット数を増やすとより複雑な入出力関係を表すことができるが、増やしすぎると学習に用いるデータの過剰な学習をしてしまい、学習に用いてないテストデータの予測性能が下がる減少する現象（過学習）が発生することが知られている。本研究では、すべての層で入力データの次元の約 6 割ということで設定した。

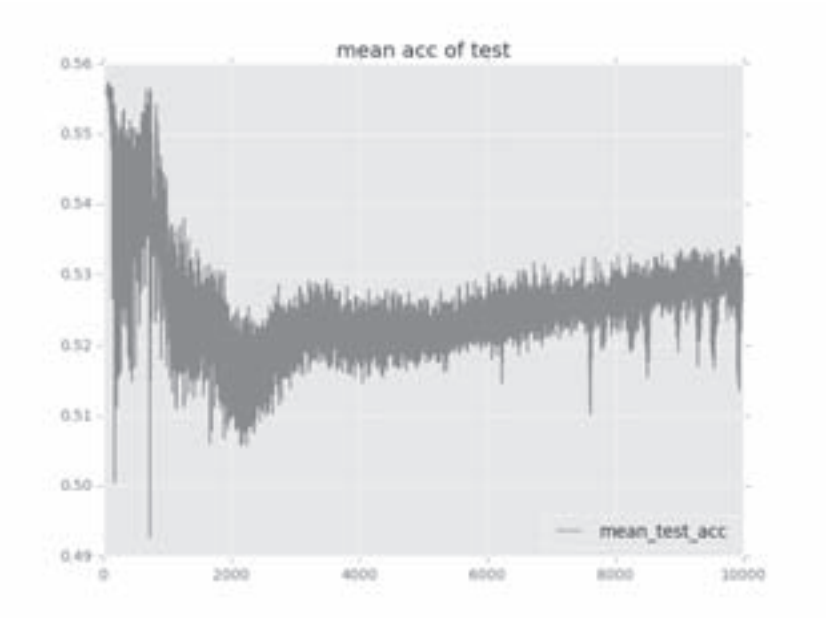

図 6: 検証データの正解率推移

各層のユニットの活性化関数は中間層の各ユニットでは tanh()（双曲線正接）を用い、出力層の 2 ユニットでは式 (4) で表されるシグモイド関数を用いる。

データは各変数ごとに以下の式でそれぞれ 0 から 1 の範囲となるよう標準化する。これは他の変数と比較し絶対値の大きなものがあると、その変数の影響を強く受け、他の変数の変化による出力層のユニットの出力への影響が打ち消されてしまうためである。入力の 1 つの変数を $\boldsymbol{v} = (v_1, v_2, \ldots, v_N)$ とし、$\max(), \min()$ をそれぞれベクトルの要素の最大値、最小値を求める関数とし、$\boldsymbol{v}' = (v'_1, v'_2, \ldots, v'_N)$ を標準化後の変数のベクトルとすると次のようになる。

$$v'_m = \frac{v_m - \min(\boldsymbol{v})}{R}, \tag{27}$$

$$R = \max(\boldsymbol{v}) - \min(\boldsymbol{v}). \tag{28}$$

これにより各変数の値の範囲は $0 \sim 1$ となる。

また、学習の際のデータの与え方はすべてのデータの誤差を計算した上でウェイト修正項を計算する一括学習ではなく、1 日のサンプル数と同じ 295 のずつ与える確率的学習法をもちいることとした。ただし、295 サンプルは 1 日のデータごとではなく、すべての学習に用いるデータをランダムに並び替えた上で 295 ずつ取り出して与えている。並べ替えは学習の 1 ステップごとに実行し直している。学習手法は Adam を用いている。

4　シミュレーション結果

図 5 には学習プロセスでの学習データに対するネットワーク出力の正解率の変化を示している。正解率は学習に用いる全データに対する、ネットワーク出力による分類が正しいサンプル数の率を

	下がる or 変わらない（実測）	上がる（実測）
下がる or 変わらない（予測）	6381	1744
上がる（予測）	1495	5080

表 1: 学習データの正誤表

	down	up
sell	143	68
buy	40	44

表 2: 1 日目

	down	up
sell	112	77
buy	62	43

表 3: 2 日目

	down	up
sell	82	83
buy	69	60

表 4: 3 日目

	down	up
sell	91	75
buy	69	59

表 5: 4 日目

	down	up
sell	126	75
buy	38	55

表 6: 5 日目

	down	up
sell	111	83
buy	57	43

表 7: 6 日目

	down	up
sell	97	93
buy	49	55

表 8: 7 日目

	down	up
sell	121	81
buy	52	40

表 9: 8 日目

	down	up
sell	109	81
buy	59	45

表 10: 9 日目

	down	up
sell	107	69
buy	73	45

表 11: 10 日目

	down	up
sell	106	82
buy	57	49

表 12: 11 日目

	down	up
sell	94	96
buy	50	54

表 13: 12 日目

用いる。学習データについて正解率が 1 ということはすべての学習データを正しく分類できる学習ができているということとなる。図 5 の x 軸は学習ステップ数で、y 軸が正解率を表している。学習データをすべて適用した段階で学習ステップが進むこととし、この図では 10,000 ステップ適用の各段階の正解率である。ただし、この図は積算自己符号器のウェイト学習後のフィードフォワードネットワーク全体での正解率を示している。この図からわかるように、学習ステップ初期ではおよそ 54%の正解率であったものが、最終的には 78.0%まで学習が進んでいることが分かる。中間層の数や各層のユニット数を増やすと 100%に近づけることはできるが、この場合過学習となり、検証データの正解率は低くなる場合もある。ところどころ正解率が大幅に下がっている箇所があるが、これは正解率の更新があまり進まない箇所での Adam による探索の影響である。　　図 6 には学習に用いていないテストデータの予測の正解率の推移を示している。図の見方は学習データの正解率の図 5 と同様である。学習ステップ初期で正解率 55%を超える箇所もあるが、これは学習進行上（適切なウェイトへの遷移途中）での誤差変動と考えられ、その後 2,000 ステップ辺りから最終ステップまで正解率は上昇傾向にあり、10,000 ステップ目での正解率は 53.6%となっている。学習データと比べると最終的正解率は低い値となっているが、50%を数パーセント超えた値となっている。

表 1 には、最終的な学習データの予測の正誤表を示している。列方向が実際の値、つまり 3 分後に株価が上がるか、下がる（変わらない含む）かを表し、行方向がディープラーニングによる出力（予測）を示している。実際に上がるケースでの正解率は 74.4%、下がるケースでの正解率は 81.0%となっている。

表 2 から表 13 にはには、検証用データの 1 日ごとの上がり、下がり（変わらない含む）の予測の正誤表を示している。表の見方は表 1 の学習データの場合と同じであるが、列方向それぞれが下がる（変わらない含む）（実測）を down と表記したものと上がる（実測）を up と表記したもの、行方向はそれぞれ予測で下がる（変わらない含む）を sell、上がるを buy と省略表記している。全体的な

傾向としては下がる（変わらない）際の予測が全体で65.8%、上がる際の予測が全体で38.1%となっていて、上がる際の予測よりも下がる際の予測がよくなっている。これは株価の過去の変動を見て上がることを予測することよりも、変動を見て下がることを予測する方が予測（学習）が容易であることを示していると考えられる。

この検証データで、上がるという予測の際は1株現物購入し、下がる（変わらない）の予測の際は1株空売りをし、それぞれ3分後に決済する取引を繰り返すと、12日間それぞれの日の損益額と損益率（カッコ内の数値）は、1日目：502円（2.8%）、2日目：195円（1.1%）、3日目：-170円（-0.9%）、4日目：-47円（-0.3%）、5日目：107円（0.6%）、6日目：51円（0.3%）、7日目：-132円（-0.7%）、8日目：70円（0.4%）、9日目：131円（0.7%）、10日目：-39円（-0.2%）、11日目：44円（0.2%）、12日目：154円（0.9%）となり、12日間のトータル損益額、損益率はそれぞれ866円、4.8%となった。　ただし、損益率を計算する際は、投資額をソフトバンク株1株をおおよそ6,000円とし、3分間で必要な投資額を3連続して1株を購入した額（最大投資額）のおおよその値である18,000円として計算した目安の値である。　これらの損益額や率はどの期間の株価を学習データ、検証データとするかや、ディープラーニングのパラメータや学習方法に影響を受けることとなるので、設定によっては結果は大きく変わってくると考えられる。ただし、損益額には取引にかかる手数料は考慮していないので、手数料を考慮するとさらなる正解率の改善が必要となると考えられる。

5　むすび

本研究では、近年研究が急速に進んできたディープラーニングの株式投資への応用の可能性について分析を行った。今回の利用株価期間、においては投資結果は良好となったが、今後は勝敗率の向上、用いる株価設定など様々な面からの検証を続けていきたい。

参考文献

[1] 岡谷貴之, 深層学習, 講談社,2015.

[2] Rosenblatt F.,“The Perceptron: A probabilistic model for information storage and organization in the brain”,Psychological Review, 65, 386-408, 1958.

[3] Minsky M. and Papert S.,Perceptrons: An Introduction to Computational Geometry. MIT Press, MA,1969.

[4] Rumelhart D. E. , Hinton G. E. and Williams R. J. ,“Learning internal representations by error propagation. Parallel Distributed Processing”, 1, MIT Press, MA, 318-362, 1986.

[5] Diederik P. K. , Jimmy L. B., “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION”, conference paper at ICLR 2015,1-15,2015.

[6] “A Powerful, Flexible, and Intuitive Framework of Neural Networks”,http://chainer.org/.

[7] 神嶌敏弘（編）, 麻生英樹, 安田宗樹　他（著）, 深層学習, 近代科学社, 2015.