LSTMを用いた株価指数の予測

18PA910

はじめに

従来型の時系列予測の研究に関しては主に統計分析手法と計量経済学モデルを用いて行っているが、1980年代以降、金融市場の分析に機械学習の手法が徐々に適用されるようになった。しかし、複雑な関数近似をしなければ分類や回帰ができない場合があり、従来型の機械学習手法ではうまくいかないケースも多く、このような問題に対して、ディープラーニング（深層学習）手法を使用するケースが増えている。ディープラーニングの領域の中でベースの手法はニューラルネットワークと呼ばれる機械学習手法である。しかし、通常のニューラルネットワークでは、ある層の出力は、次の層の入力に利用されるのみであり、時系列データを自然に扱うことができない。時系列データを予測するとき、過去の状態を中間層として定義したリカレント（再帰型）ニューラルネットワーク（以下：RNN）が用いられる場合が多い。RNNには時間を深く遡るほど勾配爆発あるいは勾配消失の問題があり、長・短期記憶（Long short-term memory、略称:LSTM）により解決された。金融市場の分析において、多くの研究はLSTMがいい予測結果が出られ、期待すべき研究方法であることを証明した。本文はLSTMを用いて世界中主要な株価指数の予測を行った。

先行研究

White（1988）[1]は、ニューラルネットワークを用いて、IBMの株収益率を予測した。Atiyaなど(1997)[2]は上場企業の株収益率、売上高と利益率などの基本的な企業情報に基づいて、ニューラルネットワークを用いて、予測を行った。結果として、ニューラルネットワークの予測結果は投資者の参考になる。金融市場において、RNNの研究に関しては、Xieなど(2016)[3]はRNNを用いて、中国の株市場に関する実証分析を行った。RNNは投資者の参考になるよい予測結果を出した。また、LSTMの発展に関しては、HochreiterとSchmidhuber(1997)[4]は再帰型ニューラルネットワークを基づいて、LSTMを提出した。Gers&Schmidhuber（2000）[5][6]は忘却ゲートと覗き穴結合（peephole connection）をLSTMに導入し、LSTMを改良した。MaknickieneとMaknickas(2012)[7]はLSTMを用いて、為替市場のユーロ／ドルを予測し、取引戦略を立ってた。結果としてLSTMの予測に基づいた戦略は実行可能である。Murtaza　Roondiwalaなど(2015)[8]はLSTMを用いて、NIFTY50の収益率を予測した。LSTMはいい予測能力を持っていることを証明した。松井藤五郎と汐月智也(2017)[9]はLSTMを用いて1分ごとの1分後利益率をモデル化する手法を提案した。結果は、過学習が生じ、テストデータに対する予測ができないものであった。南正太郎(2017)[10]はLSTM-RNNを用いたイベント考慮後の株価時系列予測した。使用したデータが少ないため、精度が低くなってしまう結果となったが、長期に渡る分析を行うなら、LSTMは幅広い可能性を持っていることが望ましい。

研究方法

1、RNNの消失の問題

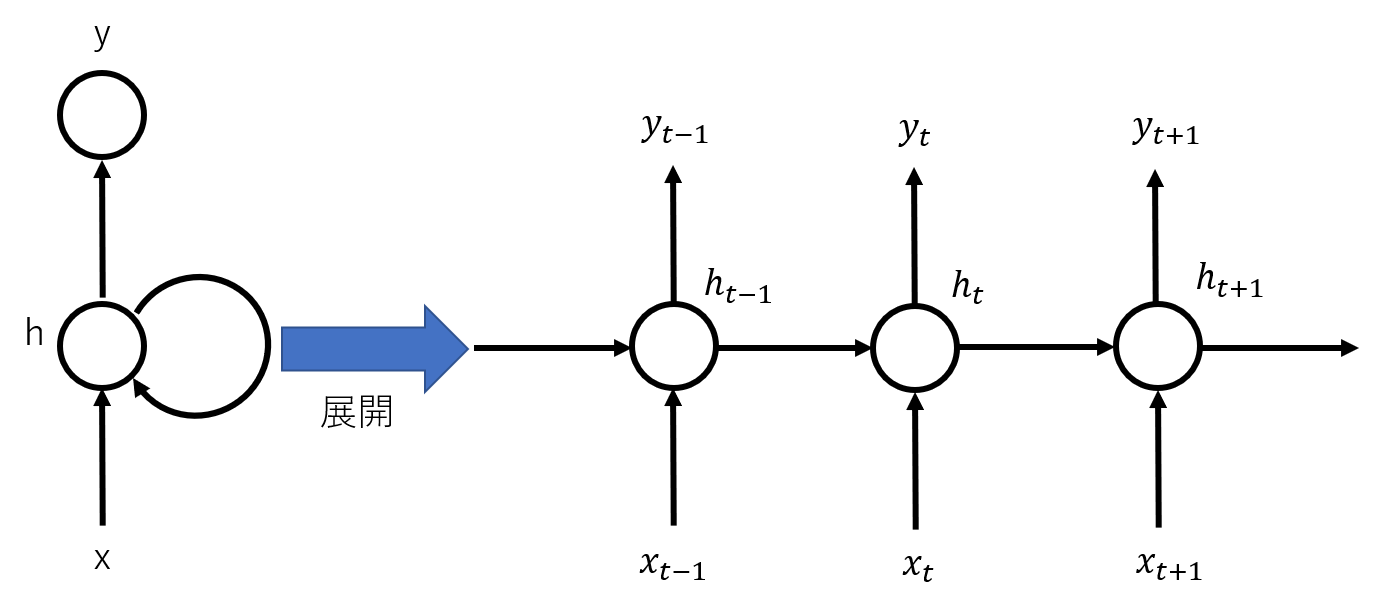


図１（RNNの概要図）

図１で示したように、RNNでは、層自体は入力層、中間層、出力層と一般的なニューラルネットワークと変わらないが、時刻tにおけるに加え、時刻t-1における中間層の値を保持しておき、それも時刻tにおける中間層に伝える点が異なる。また、各層の入力として、前の層の入力のみではなく、時系列のデータポイントも入力とする。だから、RNNは時系列データの予測問題を解決することができる。

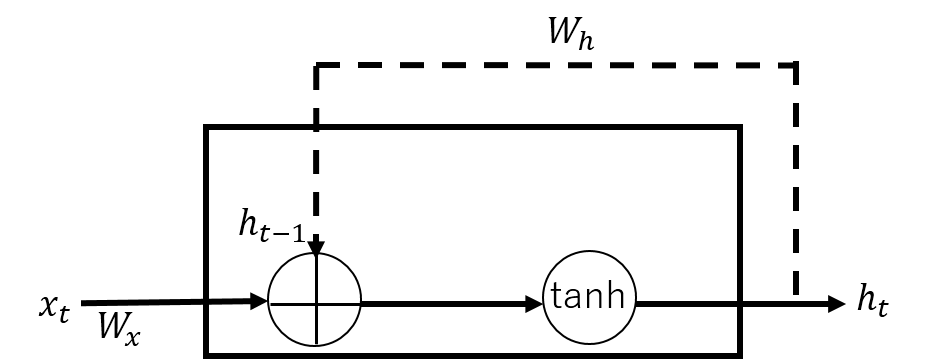


図2　従来のRNN

　　図2で示したように、時刻tの中間層の計算をする際に入力層からの入力と前の時刻の中間層の出力に重み付けを行った結果を加算したものを中間層への入力として扱う。この際にポイントが2つある。

　　一つ目は、一般的なRNNでは時刻のt=0のときには時刻t-1の中間層が存在しないため、時刻t-1の中間層からの入力が0として設定する。

　　二つ目は、入力層から中間層への重み、中間層から中間層への重み、中間層からその上の層への重みは別の時間軸関でそれぞれ同じものを用いる。すなわち、一つのニューラルネットワークを構築して、その中間層の計算にとある特定の重みを用いて時間展開をしていくということになる。その際の活性化関数としてはよくtanhが用いられる。

　　そのため、ある時刻tにおける中間層からの出力を、時刻tにおける入力、時刻入力層から中間層への重みとバイアス、中間層から中間層への重みを用いて書くと以下のようになる。

これから、RNNの勾配を計算する

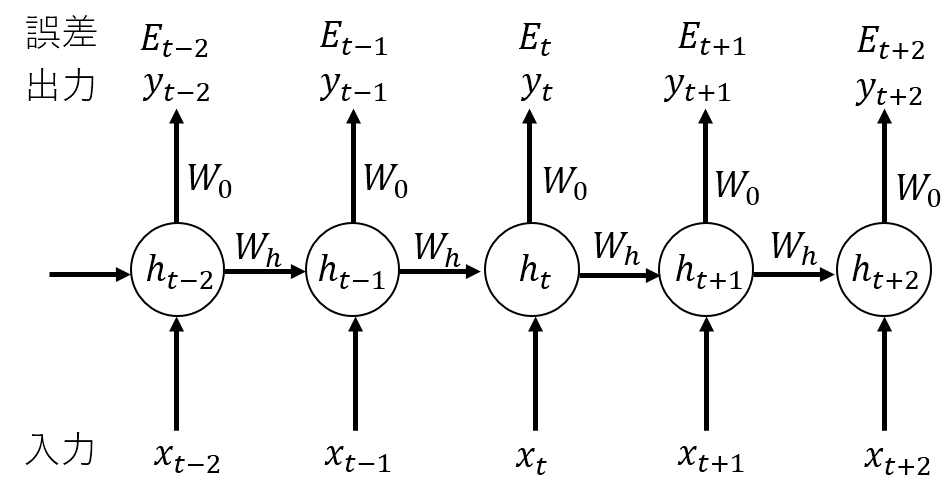


図3（RNNの展開図）

はt時刻の入力

はt時刻の中間層の出力

は誤差

は入力層から中間層の重み

は中間層同士の重み

は中間層から出力層への重み

RNNの勾配計算の手法としてメジャーなものはBack Propagation Through Time(BPTT)である。

RNNでは各時刻が重みを共有して、中間層から出力層まで

t时刻における誤差関数はmean square error(MSE)と定義される。

だから、時刻tにおける誤差をとすると、合計誤差は次のように表すことができる。

これから、誤差関数の値Eを最小になるために、すべての重み、、、についてそれぞれの勾配を計算する。

まずは、連鎖律を用いると、中間層から出力層への重みに関する誤差関数の偏微分は以下のようになる。

次は、中間層同士の重みについての勾配を計算すると以下のようになる。

はt時刻の中間層の出力を表すため、式（１）によると、はの関数であり、に関する計算はとに依存する。また前の時刻t-1のときに、であり、 に関する計算はとに依存する。だからを展開する必要がある。

を展開すると

そして、を展開すると、

式(8)を式(7)に代入すると

これにより、式(6)のという部分が展開された。

式(9)を式(6)に代入すると

最後に、入力層から中間層への重みに関する誤差関数の偏微分を計算する。の計算と同様に、連鎖律を用いると、重みに関する誤差関数の偏微分は以下のようになる

を展開すると、

先ほど計算した勾配の特性上、普通のRNNは長時間のデータを意識した学習をすることができないという問題が起こる。

重みとに関する誤差関数の偏微分との計算に関しては、時刻tにおける計算するだけではなくて、時刻kからtまで、すべての時刻を合わせて計算する必要があるから、また、総乗の部分が時間に依存するため、時刻jと時刻kの間が大きく離れた場合は勾配消失の問題が発生する。

なぜなら、の計算結果は

tanh関数の導関数の値の範囲がはであるため

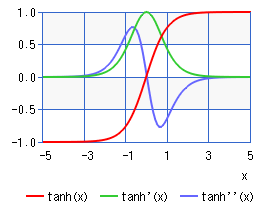


図4 tanh関数のグラフ

勾配消失が発生する原因は

のとき、もしjとkの間の距離が大きすぎると、連乗も多すぎて、0に近づき、勾配消失が発生する。

勾配消失が起こると、ある程度先の時刻以降の誤差は全く考慮しないで重みを更新するという現象が起こっている。実際には4-5時刻程先の誤差を考慮した勾配を計算するとほぼ0になる。

2. 改良したRNN―LSTM

LSTMは、Hochreiter&Schmidhuber(1997)により提出され、RNNの中間層出力に対して、記憶期間の長さの考え方を導入することにより、遠い過去の出力の影響を保持することを可能にしている。

LSTMは勾配爆発問題と勾配消失問題をCEC（Constant　Error　Carousel）という概念を導入するにより解決することができる。図5の中心にあるCECは、受け取った値をそのまま過去の値として保持し、次の時刻に伝える。すなわち、破線の矢印が時間を遡った伝播を、+印のノードが値の掛け合わせを表す。例えば、前の時刻t-1における出力と今の入力の重み付き線形和をそのまま通すではなく、前の時刻t-1におけるCECの値を加算して出力される。

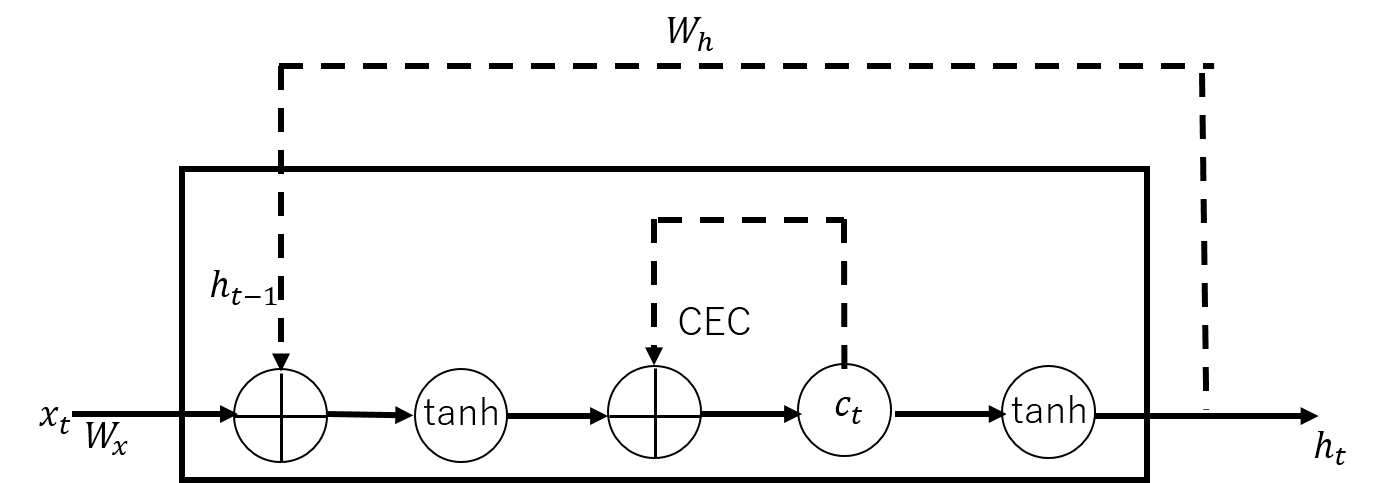


図5　CECの導入

としておく

全体としては必ず１より大きな値になるため、勾配消失が回避された。

CECを導入することにより、過去の入力情報をすべて記憶し、過去を遡っても誤差を逆伝播できるようになったが、時系列データを学習するうえで、もう1つ大きな問題がある。時系列データを入力で受け取る場合は、時間依存性がある信号を受け取ったときは重みを大きくし活性化すべき一方、依存性がない信号を受け取ったときは重みを小さくし非活性のままであるべきである。しかし実際は、ニューロンが同じ重みでつながっている限り、両者はお互いに打ち消し合う重みの更新となってしまうので、特に長期依存性の学習がうまくできないことに繋がる。

この問題を解決するためには、依存性のある信号を受け取ったときのみ活性化し、それ以外では依存性のありそうな情報を内部で保持しておく機構が必要になる。後者はCECにより実現できるが、前者の問題は、必要になったタイミングでのみ信号を伝播し、それ以外では信号を遮断するゲートのような存在を取り入れるべきである。ゲートの導入することによって各時刻ごとに入出力を制御することが可能になる。

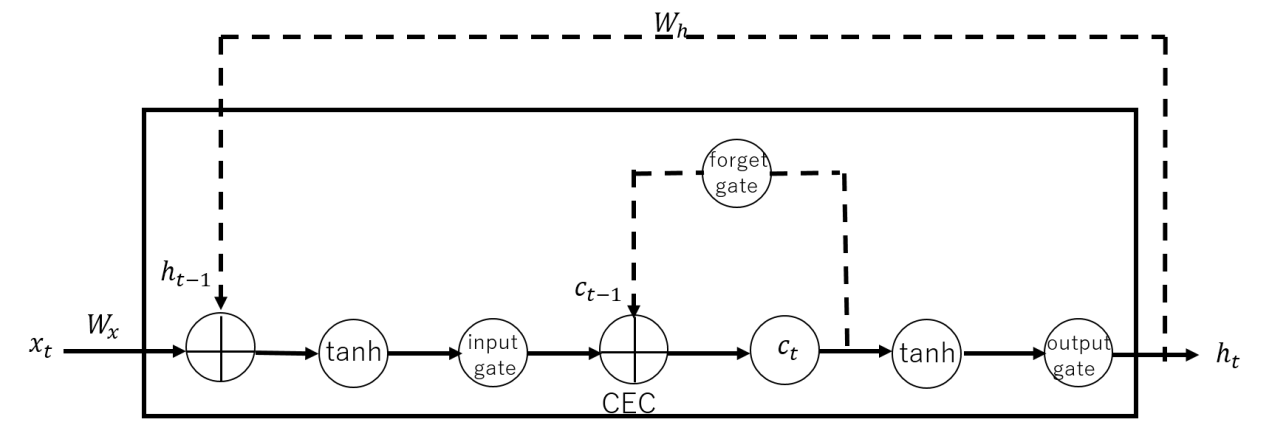


図6　ゲートの導入

ゲートはLSTMが発表された1997年は入力ゲートと出力ゲートの二つだけであったが、1999年にはGers & Schmidhuberは入力系列のパターンが変わった時にCECの状態を一気に変更できるように忘却ゲートを導入した。図6のようにCECの入力部分に入力ゲート、出力部分に出力ゲートを導入することで、入出力ともに過去の情報が必要になったタイミングでのみゲートを開け信号を伝播し、それ以外はゲートを閉じておくことで過去の情報を保持しておくことが可能になる。また、忘却ゲートに関してはCECからの誤差を受け取ることで、必要なタイミングでCECに記憶された値を忘れ去る機能を果す。

しかし、ゲートの制御に用いるのは時刻tにおける入力層の値がおよび時刻t-1における中間層の値であり、制御すべきCEC自身が保持している値は用いられていない。一見するとを制御に用いていることによりCECの状態が反映されているように思われるが、あくまでもLSTMブロックの出力は出力ゲートに依存しているので、仮に出力ゲートがずっと閉じている場合、どのゲートもCECにアクセスすることができず、CECの状態を見ることができないという問題が発生する。

この問題を解決するためにGers & Schmidhuber, 00は覗き穴結合（peephole connection）を導入した。図7の青い線で表される。これはCECから各ゲートをつなぐもので、これによりCECの状態を各ゲートに伝えることができるようになる。

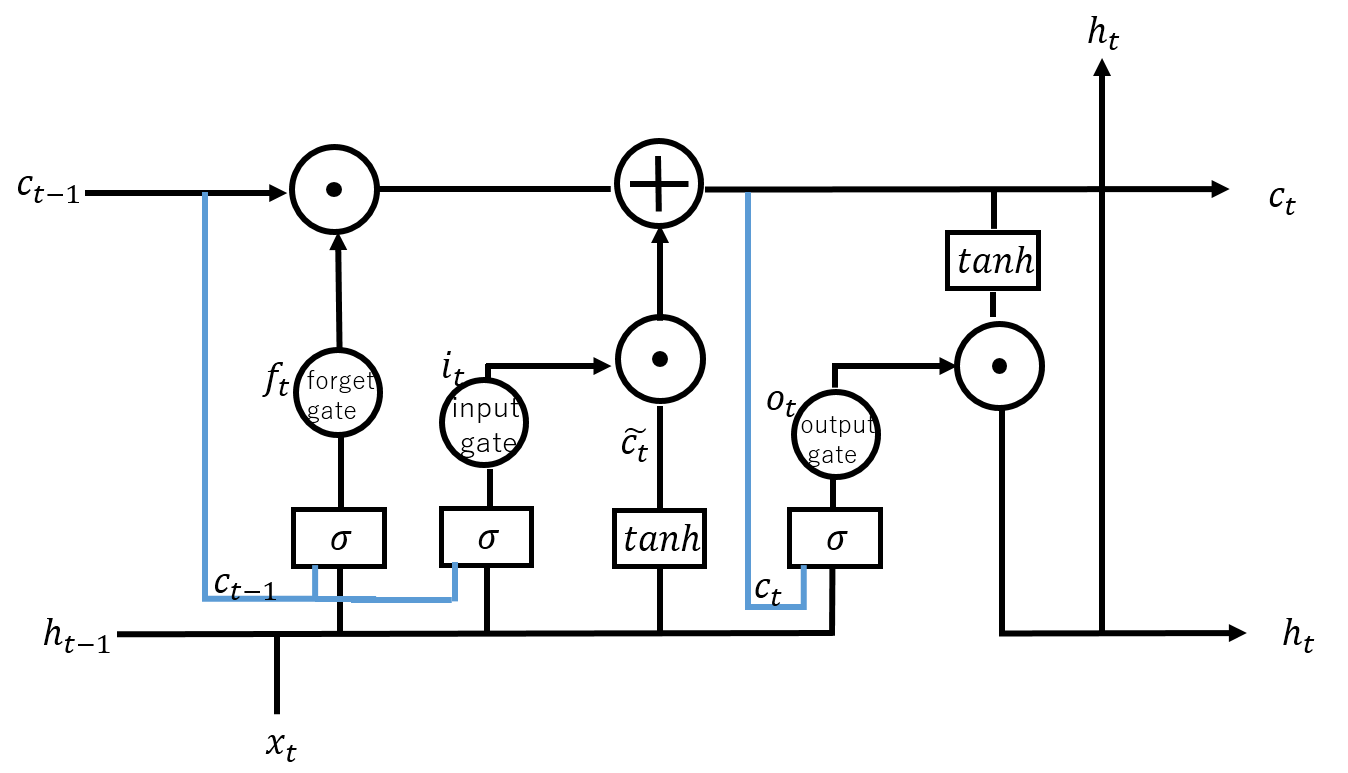


図7　peephole connectionを導入したLSTM

図の中の⊙はアダマール積であり、青い線はpeephole connectionである。

具体的にそれぞれのゲートの式は以下のようになる。

入力ゲート

入力データ

出力ゲート

忘却ゲート

、、の計算に関してはそれぞれt時刻における入力と前の時刻における中間層の出力の重み付き線形和にシグモイド関数を作用し、0.0～1.0の間の値に調整された。ゲートの値が1.0に近い場合はそれぞれの入力を通過させ、0.0に近い場合はシャットダウンする。

また、はtanh関数を作用されることにより－1.0～1.0の間の値になるように調整された。はただの入力データであり、ゲートを制御するわけではない。

これから、忘却ゲートを作用させた前の時刻のCECの値と入力ゲートを作用させた入力を時刻tの入力を加算してCECを更新する

それに活性化関数tanhを作用させた上で、出力ゲートを作用させて中間層出力とする。

⊙tanh()

最後の結果の出力は

=

これがLSTMの仕組みである。

これから、LSTMの勾配を計算する。

、、、を定義する。すると、これらを

とする

とする

とする

とする

という形でまとめる。

こうしておくことで、を求めればよいことになるので、

を計算する必要がある。

よって：

これらの式にある、、、は、

とに関しては、

よって：

これで、勾配が計算された。

データに関して

　　分析対象として、S&P500、Dow30とNikkei225の一日ごとの株価を取り上げ分析を行った。

　　データの獲得に関しては、今回で使われるデータは全てfinace.yahoo.comというサイトで獲得した。獲得したデータにはopen,close,low,high,adj closeという６の種類の価格がある。そして、今回の分析に用いる変数は、Adj Close価格のみである。使用したデータの期間は2010年1月4日から2019年12月27日まで10年間のデータであり、合わせての数は2514である。全期間の7割をトレーニング期間、残りの3割がテスト期間と設定する。

　　データの前処理に関しては、まず、NA値が入っていたデータを取り除く必要がある。

　　次は、データの形を変換することである。データをLSTMに入力するためには、データを正規化する必要がある。今回の正規化で使われる方法としてはMinMaxScalerである。これはデータの中における最大値と最小値を使って正規化する方法である。

　　MinMaxScalerを用いてデータを処理することで、データの最大値が1、最小値が0となった。

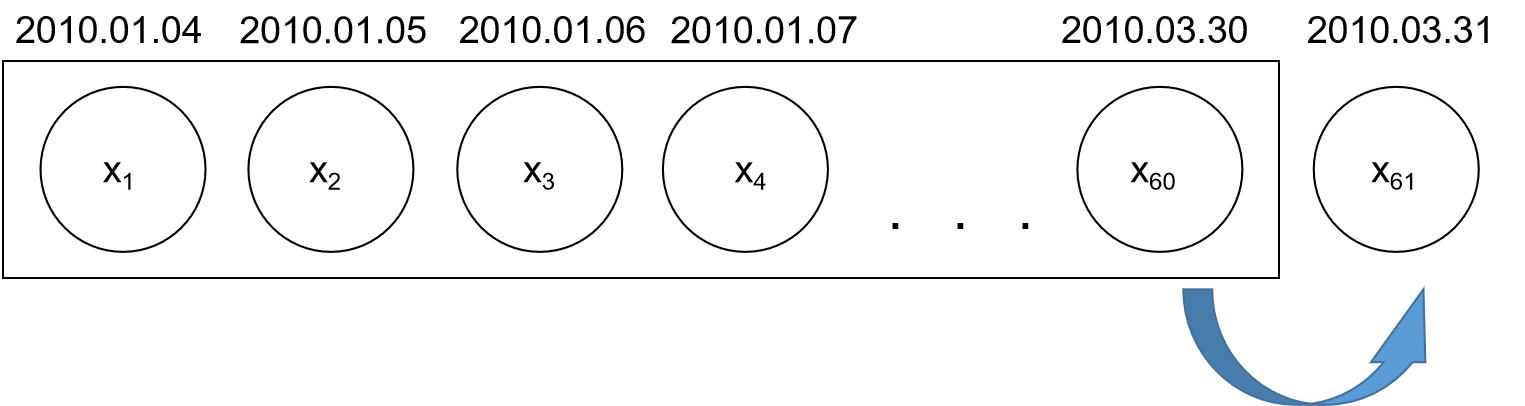
正規化した後、連続した1日のAdj Close価格のデータを X\_trainとy\_train二つのグループに分ける。毎回の学習に関しては、第1個目から第n個目まで、合わせてn個のデータをX\_trainに保存し、順番にLSTMへの入力する。n 個目のAdj Close価格を入力したときに LSTM から出力された値がn+1個目のAdj Close価格の予測値となり、n+1個目のAdj Close価格を教師信号として LSTMを学習させる。例えば、2010年1月4日が第1個目となり、60日の期間を基づいて、2010年3月31日(61個目)のAdj Closeの値を予測したい場合は、この日から2010年3月30日(60個目)まで、連続した1日のAdj Closeの値を用意し、1個目における1日後Adj Close価格から順に入力する。60個目のAdj Closeの値を入力したときの出力が2010年3月31日(61個目)のAdj Closeの値となるように、61個目のAdj Closeの値を教師信号として LSTM を学習させる。

図8

LSTMの実装

　　南正太郎(2017)[10]のLSTMモデルを参考にして、本文のLSTMモデルを設定した。学習時には、トレーニングデータのX\_trainの期間をそれぞれ60、90、120の3つの長さで設定する。また、batch\_sizeを32で設定し、epoch数を50、100、150、200の4つのパターンで設定し、つまりそれぞれ50、100、150、200回の反復処理を行うことになる。今回の研究では、まずは、アメリアのS&P500株価指数を使い、期間とepochの組み合わせの12パターンで計算し、最適な予測期間とepochの組み合わせのパターンを見つけ出す。次に、日本のNikkei225(日経平均株価指数)とアメリアのDow30株価指数を使い、前の段階で見つけ出したパターンの検証を行う。

　　隠れ層のユニット数は50、LSTM層は4とする。学習のフレームワークとしては、フロントエンドに Keras、バックエンドには Tensorflowを使用して実装している。確率的勾配降下法の最適化を行う上で、Adam(Adaptive Moment Estimation) により学習率の設定を行っている。

　　出力に関しては、図9のように、many to oneが使われる。例としては、60日の期間に基づいて予測する場合である。

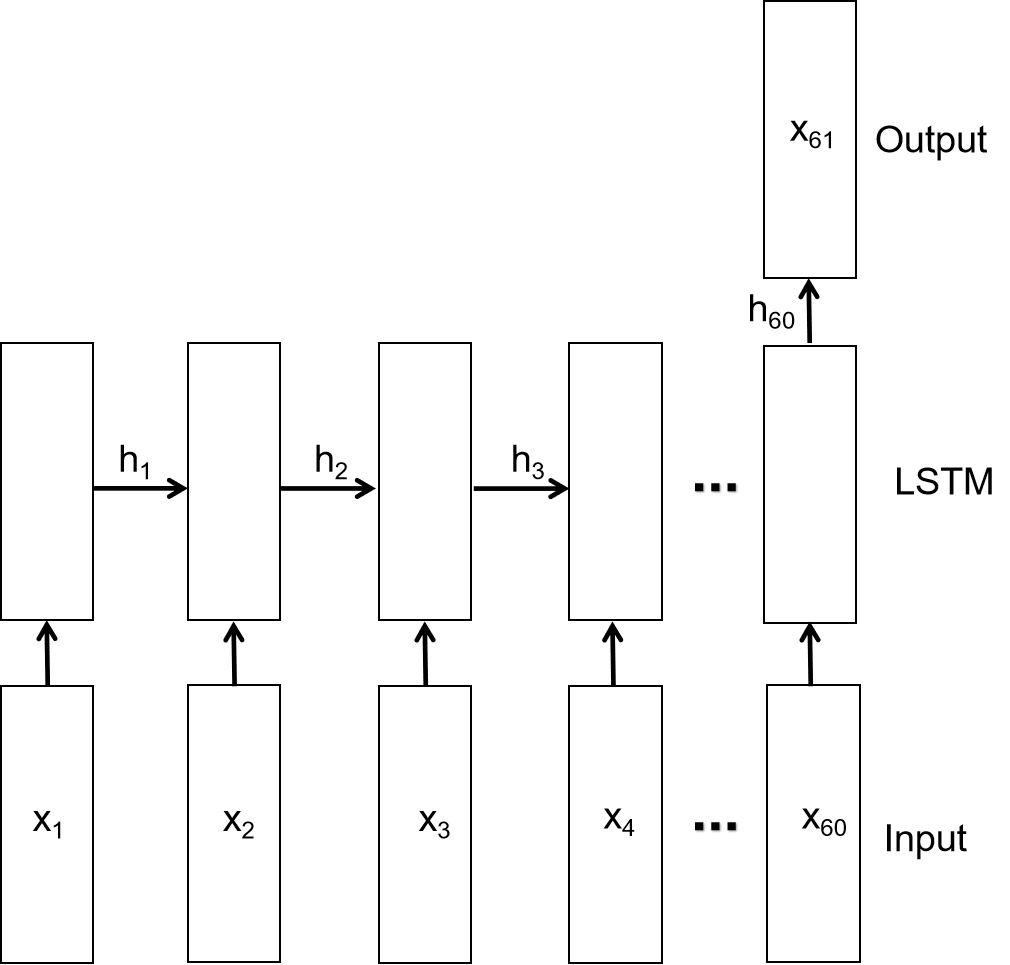


図9

研究結果

60日、90日と120日の期間に基づいて、S&P500を予測するとき、Epochがそれぞれ50、100、150と200の場合の結果は図10、図11と図12である。



図10　60日の期間/epoch=50,100,150,200



図11　90日の期間/epoch=50,100,150,200

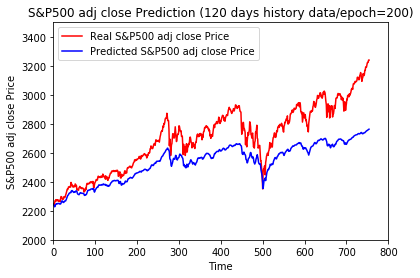
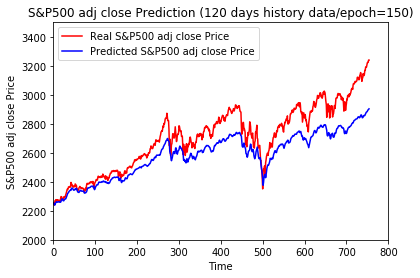
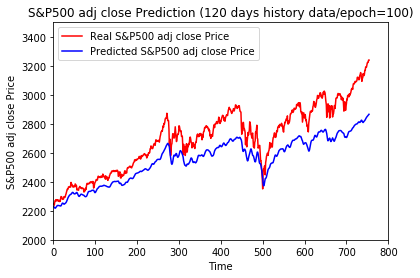


図12　120日の期間/epoch=50,100,150,200

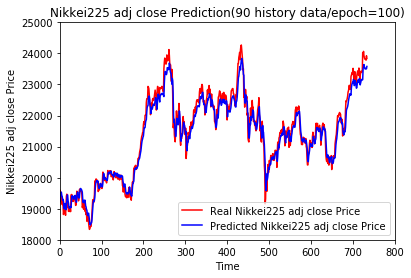
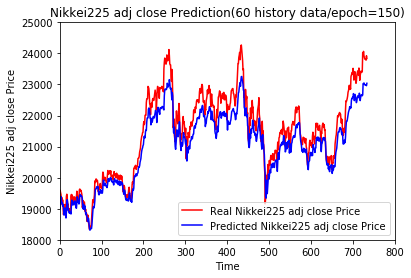
　　LSTMによる株価の予測結果は真の株価の変化の流れとはあまり変わりがないことがわかる。

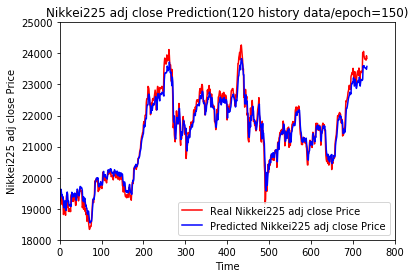
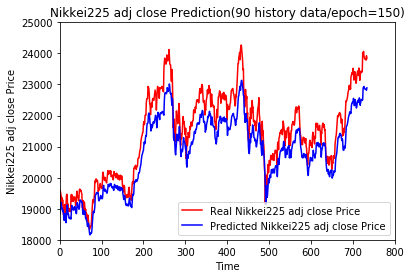
図10で示したように、同じ60日の期間に基づいて予測した結果中で、epoch=150の場合の予測結果が一番当てはまりがいいことが分かる。epoch=200の予測結果が逆に悪くなった原因の一つとしては過学習が発生した可能性がある。図11と図12を見ると、同じ期間に基づいて予測した結果の中ではepoch=100とepoch=150の間で、あまり変わらないのたが、timeが0から100までのところで、epoch=150の方が当てはまる予測結果がよいことがわかる。

図10、11、12の中で合計12個の図を総合的に見ると60日の期間epoch=150、90日の期間epoch=100,150と120日の期間epoch=150という4つのパターンの予測結果は他のパターンより当てはまる結果がいいことがわかる。

これから、検証の段階に入り、Nikkei225株価指数とDow30株価指数を使い、この4つのパターンで検証を行う。また、Nikkei225株価指数とDow30株価指数を予測することにあたって、この4つのパターン中から、それぞれ最も当てはまる結果がよいパターンを見つけ出す。

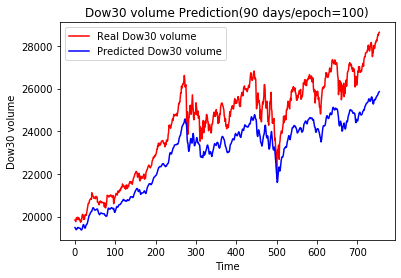
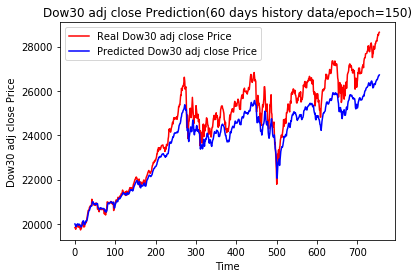
Nikkei225株価指数を使い、その4つのパターンで予測を行う。

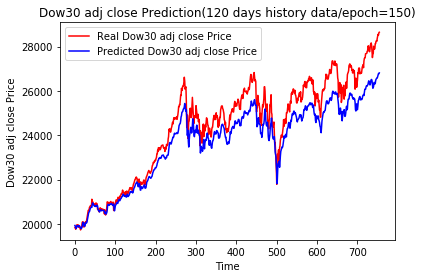
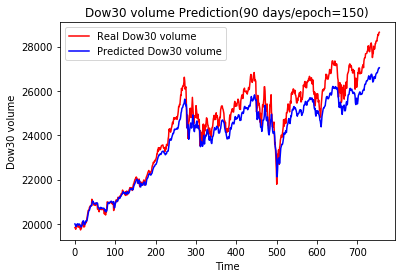




　　Nekkei225の予測結果からみると90日の期間に基づいて、epoch=100のパターンの予測結果と120日の期間に基づいて、epoch=150のパターンの予測結果がほぼ真の株価と一致し、この2つのパターンの予測結果が最もよく当てはまることがわかる。

　　Dow30株価指数を使い、その4つのパターンで予測を行う。





　　Dow30の予測結果からみると90日の期間に基づいて、epoch=100のパターンの予測結果が最も悪いことがわかる。60日の期間に基づいてepoch=150、90日の期間に基づいてepoch=150と120日の期間に基づいてepoch=150という3つパターンの予測結果があまり変わりがなかったが、200-300のところでは、90日の期間に基づいて、epoch=150のパターンの予測結果が他の2つよりよいことがわかる。

結論

本文では、LSTMを用いて、S&P500株価指数の予測を行い、予測期間の長さとepochを設定することにより、最適な4つのパターンを見つけ出した。そして、Nikkei225、Dow30という2つの株価指数を使い、検証を行った。結果によると、株価指数を予測することにあたって、LSTMの手法では、真の株価指数とは完全に一致しなくても、株価指数の変動の流れが予測できることが明らかになった。予測期間を90日で設定し、epochを150で設定した場合の予測結果が最もよく当てはまることが明らかになった。

LSTMの手法は、金融市場の分析では幅広い応用の可能性が持っていることが明らかにした。

これからの展望

本文では、予測期間が60日、90日と120日、Epochがそれぞれ50、100、150と200を基づき、合わせて12パターンだけの予測を行った。予測期間とEpochをもっと細かく設定する余地が残っている、LSTMの手法の予測精度が上がる可能性が十分にある。

本文で使われるコード

# Part 1 - Data Preprocessing

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the training set

dataset\_total = pd.read\_csv('filename.csv')

train\_size = int(len(dataset\_total)\*0.7)

test\_size = len(dataset\_total)-train\_size

training\_set = dataset\_total.iloc[:train\_size, 5:6].values #choose the adj close column

test\_set = dataset\_total.iloc[train\_size:len(dataset\_total), 5:6].values

# Feature Scaling

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0, 1))

training\_set\_scaled = sc.fit\_transform(training\_set)

# Creating a data structure with 60 timesteps and 1 output

timesteps = 60 #we can also change the timesteps to 90 and 120

epoch = 100 # we can also change the epoch to 50,150 and 200

X\_train = []

y\_train = []

for i in range(timesteps, len(training\_set\_scaled)):

X\_train.append(training\_set\_scaled[i-timesteps:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshaping

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# Part 2 - Building the RNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

# Initialising the RNN

regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50))

regressor.add(Dropout(0.2))

# Adding the output layer

regressor.add(Dense(units = 1))

# Compiling the RNN

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

# Fitting the RNN to the Training set

regressor.fit(X\_train, y\_train, epochs = epoch, batch\_size = 32)

# Part 3 - Making the predictions and visualising the results

# Getting the real stock price

real\_stock\_price = dataset\_total.iloc[train\_size:len(dataset\_total), 5:6].values

#choose the adj close cloumn

# Getting the predicted stock price

#dataset\_total = pd.concat((dataset\_train['Open'], dataset\_test['Open']), axis = 0)

inputs=dataset\_total.iloc[len(dataset\_total)-len(test\_set)-timesteps:len(dataset\_total),5:6].valuesinputs = inputs.reshape(-1,1)

inputs = sc.transform(inputs)

X\_test = []

for i in range(timesteps, len(test\_set)+timesteps):

X\_test.append(inputs[i-timesteps:i, 0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

# Visualising the results

plt.plot(real\_stock\_price, color = 'red', label = 'Real Stock Name Stock Price')

plt.plot(predicted\_stock\_price, color = 'blue', label = 'Predicted Stock Name Stock Price')

plt.title('Stock Name Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Stock Name Stock Price')

plt.legend()

plt.show()

参考文献

[1]H.White.Economic: Prediction Using Neural Networks:the Case of IBM Daily Stock Returns.Neural Networks[J].IEEE International Conference on.1988.2(6):451-458

[2]Atiya A,Talaat N,Shaheen S.: An Efficient Stock Market Forecasting Model Using Neural Network[C].International Conference on Neural Networks.1997.4:2112-2115

[3]Xie X K,Wang H.: Recurrent Neural Network for Forecastin Stock Market Trend[J].International Conference on Computer Science,2016

[4]HOCHREITER S,SCHMIDHUBER J.:Long short-term memory[J].Neural Computarion,1997,9(8):1735-1780

[5]GERS F A,SCHMIDHUBER J,CUMMINS F.: Learning to forget:Continual prediction with LSTM[J].Neural Computation,2000,12(10):2451-2471

[6]GERS F A,SCHMIDHUBER J,CUMMINS F.: Recurrent nets that time and count.Neural Networks,2000.IJCNN 2000,Proceedings of the IEEE-INNS-ENNS International Joint Conference on,volume 3,pp.189-194,IEEE,2000

[7]MAKNICKIENE N,MAKNICKAS A.: Application of neural networks for forecastin of exchange rates and forex trading[C].The 7th International Scientific Conference"Bussiness and Management 2012".Vilnius,Lithuania:Vilnius Gediminas Technical University,2012:10-11

[8]Murtaza Roondiwala, Harshal Patel, Shraddha Varma.: Predicting Stock Prices Using LSTM[J].International Journal of Science and Research(IJSR),2015,ISSN(Online):2319-7064

[9]松井　藤五郎,汐月智也: LSTMを用いた株価変動予測[J].The 31st Annual Conference of the Japanese Society for Artificial Intelligence,2017

[10]南　正太郎: LSTM－RNNを用いたイベント考慮後の株価時系列予測[J].人工知能学会研究会資料,2017:SIG-FIN-019