https://www.datacamp.com/community/tutorials/lstm-python-stock-market

LSTM による株式市場予測

長期短期記憶（LSTM）ネットワークを Python で発見し、株式市場予測を行うためにそれらを使用する方法!

このチュートリアルでは、長期短期記憶と呼ばれる時系列モデルを使用する方法を説明します。LSTM モデルは、特に後で説明する仕様で長期記憶を保持するために強力です。このチュートリアルでは、次のトピックに取り組みます。

- why 株価の動きを予測できる理由を理解する。
- データをダウンロードする - あなたは、ヤフーの金融から収集した株式市場のデータを使用します。
- 列車テストデータを分割し、データの正規化も実行します。
- 1 ステップ先の予測に使用できるいくつかの平均化手法を適用します。
- LSTM モデルは、1 ステップ以上の予測を可能にするため、動機付けと簡単に議論します。
- 現在のデータで将来の株式市場を予測し、視覚化する

ディープラーニングやニューラルネットワークに慣れていない場合は、Python のディープラーニングコースを見てください。 それは基本だけでなく、Keras で自分でニューラルネットワークを構築する方法をカバーしています。これは、このチュートリアルで使用される TensorFlow とは異なるパッケージですが、アイデアは同じです。

時系列モデルが必要な理由

株価を正しくモデル化したいので、株式購入者は株式を購入する時期と、いつ売却して利益を上げるかを合理的に決定することができます。これが時系列モデリングの登場です。データのシーケンスの履歴を確認し、シーケンスの将来の要素がどうなるかを正しく予測できる優れた機械学習モデルが必要です。

警告: 株式市場の価格は非常に予測不可能で不安定です。これは、データに、ほぼ完全に時間をかけて株価をモデル化できる一貫したパターンがないことを意味します。私からそれを取るな、プリンストン大学のエコノミスト、バートン・マルキエルは、1973 年の著書「ランダム・ウォーク・ダウン・ウォール・ストリート」で、市場が本当に効率的で、株価が公表されるとすぐにすべての要因を反映するならば、新聞の上場でダーツを投げる目隠しされた猿は、投資専門家と同様に行うべきだと主張しています。

しかし、これは単なる確率的またはランダムなプロセスであり、機械学習の希望がないと信じないようにしましょう。少なくともデータをモデル化して、予測がデータの実際の動作と相関するように見てみましょう。つまり、将来の正確な株価価値は必要ありませんが、株価の動き(つまり、近い将来に下落が起こる場合)は必要ありません。

```
# コードを正常に実行するために、これらのすべての libaries が使用可能であることを確認します。
インポート import データ pandas_datareader
インポートマットプロットリブ.ピプロットを plt として
pd としてパンダをインポート
dt として日時をインポート
インポート urllib.request, json
をインポートする
np として numpy をインポート
tf # このコードはテンソルフロー 1.6 でテストされています tensorflow
sklearn.前処理からインポート最小マックススケール
```

データのダウンロード

次のソースからのデータを使用します。

1. アルファヴァンテージ株式 API.ただし、開始する前に、API キーが必要 here になります。その後、そのキーを api_key 変数に割り当てることができます `api_key`。このチュートリアルでは、アメリカン航空の株式の 20 年の履歴データを取得します。オプションとして、過去の市場データを扱うベストプラクティスについては、この株式 API スターターガイドを参照してください。
2. このページのデータを使用します。zip ファイルの *Stocks* フォルダをプロジェクトのホーム フォルダにコピーする必要があります。

株価は、いくつかの異なる味で来ます。彼らは、

- オープン: その日の開店株価
- 閉店: 当日の終値株価

- 高: データの最高株価
- 低:その日の最低株価

アルファバンテージからのデータの取得

まず、アルファヴァンテージからデータを読み込みます。アメリカン航空の株式市場価格を利用して予測を行うため、ティッカーを「AAL」に設定します。さらに、url_string過去 20 年以内にアメリカン航空のすべての株式市場データを含む JSON ファイルを返すurl_string と、file_to_saveデータを保存するファイルとなる file_to_save を定義します。事前に定義したティッカー変数を使用して、このファイルに名前を付けます。

次に、条件を指定します: まだデータを保存していない場合は、先に進み、url_stringで設定した URL からデータを取得します。日付、低、高、ボリューム、クローズ、オープン値をパンダの DataFrame df に保存df し、file_to_save に保存しますfile_to_save。ただし、データがすでに存在する場合は、CSV から読み込むだけです。

Kaggle からのデータの取得

Kaggle 上で見つかったデータは csv ファイルのコレクションであり、前処理を行う必要はありませんので、Pandas DataFrame に直接データをロードできます。

```
data_source = 'kaggle'  # アルファバンテージまたはカグル

data_source == 'αvantage' の場合:
    #
==========================================================================================================
======================

    api_key = '<API キー>'

    # アメリカン航空の株式市場価格
    ティッカー = "AAL"

    過去 20 年間の AAL のすべての株式市場データを含む#JSON ファイル
    url_string = "https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=%s&outputsize=full&apikey=%s"%(ティッカー,api_key)

    # このファイルにデータを保存
```

```
    file_to_save = 'stock_market_data-%s.csv'%ティッカー


    # まだデータを保存していない場合は、
    #先に行くと URL からデータをつかむ
    #そして、パンダのデータフレームに日付、低、高、ボリューム、閉じる、オープン値を保存
os.path.exists(file_to_save でない場合):
urllib.request.urlopen(url_string) を url として使用:
                データ = json.ロード(url.read())デコード())
                #株式市場データを抽出
                データ = データ][時系列（毎日)']
                df = pd.データフレーム(列=[日付]、'低'、'高'、'閉じる'、'オープン'])
データの k,v の場合。
                    日付 - dt.datetime.strptime (k, '%Y-%m-%')
                    data_row = [date.date()]、フロー(v['3.低])、フロート(v[2.高]))
                                フロート(v['4. 閉じる']))、フロート(v['1. オープン'])
                    df.loc[-1,:] = data_row
                    df.インデックス = df.index + 1
        印刷（'データは保存されました：%s'%file_to_save)
        df.to_csv(file_to_save)


    #データがすでにそこにある場合は、CSV からロードするだけです
    それ以外:
        印刷('ファイルは既に存在します。CSV' からデータを読み込む)
        df = pd.read_csv(file_to_save)


それ以外:


    #
==========
==========
    #あなたは HP のデータを使用します。他のデータを自由に試してみてください。
    #しかし、そうしている間、十分な大きさのデータセットを持ち、またデータの正規化に注意してください
```

```
df =,pd.read_csv('株式'、'hpq.us.txt''、,'hpq.us.txt'区切り記号=',',、ユースコルズ=[日付]、[オープン]、'高'、'低'、'閉じる'])

印刷(Kaggle リポジトリからデータを読み込んだ))

保存されたデータ：stock_market_data-AAL.csv
```

データ探索

ここで、データフレームに収集したデータを印刷します。また、データの順序は時系列モデリングにおいて重要であるため、データが日付順に並べ替えられていることも確認する必要があります。

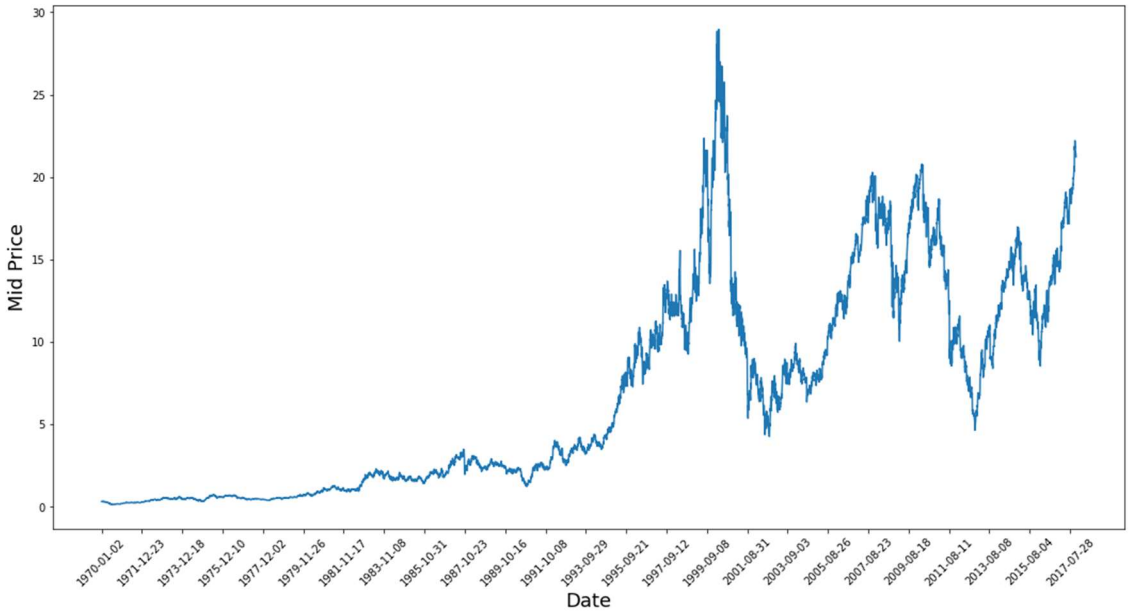
```
# 日付順にデータフレームを並べ替える

df = df.sort_values('日付')


#結果をダブルチェック

df.ヘッド()
```

	日付	開く	高	低
0	1970-01-02	0.30627	0.30627	0.30627
1	1970-01-05	0.30627	0.31768	0.30627
2	1970-01-06	0.31385	0.31385	0.30996
3	1970-01-07	0.31385	0.31385	0.31385
4	1970-01-08	0.31385	0.31768	0.31385

データの視覚化

次に、どのような種類のデータを持っているのでしょうか。さまざまなパターンを持つデータが時間の経過とともに発生する場合。

```
plt.figure(figsize = (18,9))

plt.plot(]+df['High'])/2.0)範囲(df.shape[0])(df['低'

500],rotation=plt.xticks(範囲()) 45].loc[::00,df.shape)0,500,df['日付'],

plt.xlabel('日付'、フォントサイズ=18))

plt.ylabel('ミッドプライス'、フォントサイズ=18))

plt.show()
```



このグラフはすでに多くのことを述べています。私が他の会社よりもこの会社を選んだ具体的な理由は、このグラフが時間の経過とともに株価の異なる行動で破裂しているということです。これにより、学習がより堅牢になるだけでなく、さまざまな状況に対する予測の良さをテストするための変更が与えられます。

もう一つ注目しなければならないのは、2017 年に近い値がはるかに高く、1970 年代に近い値よりも変動していることです。したがって、データが期間全体で同様の値範囲で動作することを確認する必要があります。データ*の正規化*フェーズで、この処理を行います。

トレーニングセットとテストセットへのデータの分割

あなたは、一日に最高と最低記録価格の平均を取ることによって計算された中間価格を使用します.

```
#最初に最高と最低から中間価格を計算
high_prices = df.loc[:,'高'.as_matrix()
low_prices = df.loc[:,'低'.as_matrix()
mid_prices = (high_prices+low_prices)/2.0
```

これで、トレーニング データとテスト データを分割できます。トレーニングデータは時系列の最初の 11,000 データポイントとなり、残りはテストデータになります。

```
train_data = mid_prices[:11000]]
test_data = mid_prices[11000:]
```

データの正規化

次に、データを正規化するスケーラを定義する必要があります。MinMaxScalar は、すべてのデータを 0 と 1 の領域にスケールします。また、トレーニングとテストデータの形状を[data_size、num_features]に変更することもできます。

```
# データを 0 から 1 の間にスケーリングする
#スケーリングが覚えているとき!トレーニング データに関してテストデータとトレーニング データの両方を正規化する
#テストデータへのアクセス権を持つべきではないので
スケーラー = 最小マックススケーラー()
train_data = train_data.reshape(-1,1)
test_data = test_data.reshape(-1,1)
```

以前に行った観測値、つまり、データの異なる期間の値の範囲が異なるために、完全なシリーズをウィンドウに分割してデータを正規化します。これを行わない場合、以前のデータは 0 に近い値になり、学習プロセスに多くの価値が追加されません。ここでは、2500 のウィンドウサイズを選択します。

ヒント: ウィンドウ サイズを選択するときは、ウィンドウ正規化を実行すると、各ウィンドウが個別に正規化されるため、各ウィンドウの一番最後にブレークが発生する可能性があるため、小さすぎないようにしてください。

この例では、4 つのデータ ポイントがこの影響を受けます。しかし、11,000 のデータポイントを持っている場合、4 つのポイントは問題を引き起こしません

```
#トレーニングデータとスムーズなデータでスケーラーを訓練
smoothing_window_size = 2500
範囲のディの場合は、0,10000,smoothing_window_size):,smoothing_window_size):
    scaler.fit(train_data[di:di+smoothing_window_size,:])
    train_data[di:di smoothing_window_size,:] : scaler.transform(train_data[の train_data:di smoothing_window_size,:])


# 残りのデータの最後のビットを正規化します。
scaler.fit(train_data[di+smoothing_window_size:,:])
train_data[di+smoothing_window_size::] = scaler.transform(train_data[di smoothing_window_size::])
```

[data_size]の形状に戻ってデータの形状を変更します。

```
#列車とテストデータの両方を再形成
train_data = train_data.reshape(-1)


# テストデータの正規化
test_data = scaler.transform(test_data).-1)
```

指数移動平均を使用してデータを平滑化できるようになりました。これにより、株価のデータの固有のぼろぼりを取り除き、より滑らかな曲線を生成するのに役立ちます。

トレーニング データを平滑化する必要があります。

```
#今指数移動平均平滑化を実行します
#だから、データは元のぼろぼろのデータよりも滑らかな曲線を持つことになります
EMA = 0.0
ガンマ  = 0.1
範囲の ti のための 11000):):
  EMA -範囲  -train_data[ti] - (1-ガンマ)
  train_data = EMA

#視覚化とテストの目的で使用
all_mid_data = np.連結([train_data,test_data],軸=0))
```

平均化によるワンステップ・先行予測

平均化メカニズムを使用すると、将来の株価を以前に観察された株価の平均として表すことで(多くの場合、一度先に)予測することができます。複数の時間ステップでこれを行うと、非常に悪い結果が生成される可能性があります。以下の 2 つの平均化手法を見てみましょう。標準平均化と指数移動平均。2 つのアルゴリズムによって生成された結果を定性的に評価します（目視検査）と定量的に（平均二乗誤差）の両方を評価します。

平均二乗誤差(MSE)は、一歩先の真値と予測値の間の二乗誤差を取り、すべての予測を平均することで計算できます。

標準平均

この問題をまず平均計算の問題としてモデル化してみると、この問題の難しさを理解できます。まず、将来の株式市場価格（たとえば、x_{t+1} ）を、固定サイズのウィンドウ内で以前に観測された株式市場価格の平均値として予測します（たとえば、x_{t-N}、.., x_t)(前の 100 日など)。その後、あなたはもう少し空想的な"指数移動平均"方法を試してみて、それがどれだけうまくいくかを見ます。その後、時系列予測の「聖杯」に進みます。長期短期記憶モデル。

まず、通常の平均がどのように機能するかを確認します。それはあなたが言うことです、

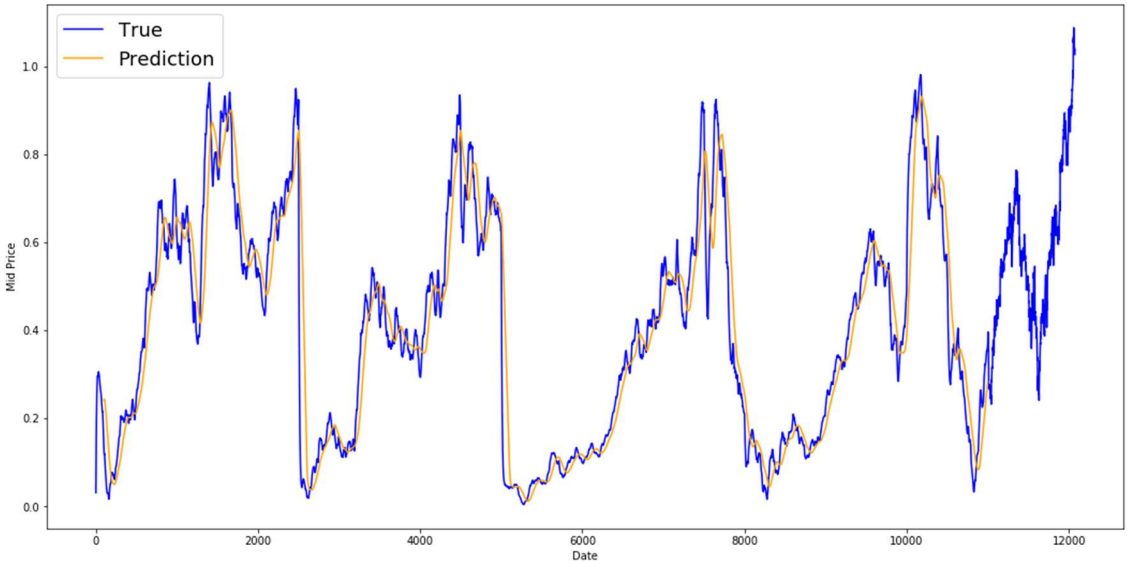
$$x_{t+1} = 1/N \sum_{i=t-N}^{t} x_i$$

つまり、t+1t+1 の予測は、t から tt-Nt-N のウィンドウ内で観察したすべての株価の平均値であると言います。

```
window_size = 100

いいえ train_data.サイズ

std_avg_predictions = []

std_avg_x = []

mse_errors = []


 in 範囲内の pred_idx(window_size,N):


pred_idx >= N の場合:

日付 = dt.datetime.strptime(k, '%Y-%m-%d')日付() + dt.timedelta (日数=1))

    それ以外:

        日付 = df.loc[pred_idx、'日付']


    std_avg_predictions.append(np.mean)(train_data[pred_idx window_size:pred_idx))

    mse_errors.append(std_avg_predictions[-1]-train_data[pred_idx]) **2)

    std_avg_x.append(日付)


印刷('標準平均の MSE エラー: %.5f'%(0.5*np.mean(mse_errors)) %(

標準平均の MSE エラー: 0.00418
```

以下の平均結果を見てみましょう。それは株式の実際の動作に非常に密接に従います。次に、より正確な 1 ステップ予測方法を見てみます。

```
plt.figure(figsize = (18,9))

plt.plot(範囲(df.shape[0])、all_mid_data、0 色='b'、ラベル='真')

plt.plot(範囲(window_size,N),std_avg_predictions,色='オレンジ'、ラベル='予測')

#plt.xticks(範囲(0,df.shape[0],50)、df[日付].loc[::50]、回転=45)

plt.xlabel('日付')

plt.ylabel('ミッドプライス')

凡例（フォントサイズ=18)

plt.show()
```



では、上記のグラフ(および MSE)は何と言っているのでしょうか?

それは非常に短い予測(1 日前)のためのモデルのそれほど悪くないようです。株価が一晩で 0 から 100 に変わらないことを考えると、この行動は賢明です。次に、指数移動平均と呼ばれる、より多くの平均化手法を見てみましょう。

指数移動平均

あなたは非常に複雑なモデルを使用して、株式市場のほぼ正確な行動を予測して、インターネット上のいくつかの記事を見たかもしれません.しかし、**注意してください!**これらは単なる錯覚であり、何か役に立つことを学ぶことによるものではありません。単純な平均化方法を使用して、その動作を複製する方法を以下に示します。

指数移動平均法では、xt+1xt+1 を次のように計算します。

- x_t+1=EMA_t= γ ×emaEMAt-1+(1- γ)xtx_t+1=EMA_t= γ ×EMAt-1+(1− γ)xt を指定すると、EMA0=0=0EMA0=0 および EMAEMA は時間の経過とともに維持する指数移動平均値です。

上記の式は基本的に t+1t+1 時間ステップから指数移動平均を計算し、それを 1 ステップ先の予測として使用します。 γ γ γ は、最新の予測が EMA に対して何を寄与するかを決定します。たとえば、 γ γ =0.1 γ =0.1 は、EMA に現在の値の 10% しか取得できません。最新のほんの一部しか取れないので、平均の非常に早い段階で見たはるかに古い値を保存することができます。以下の 1 ステップ先を予測するために使用した場合、これがどのように見えるかを確認してください。

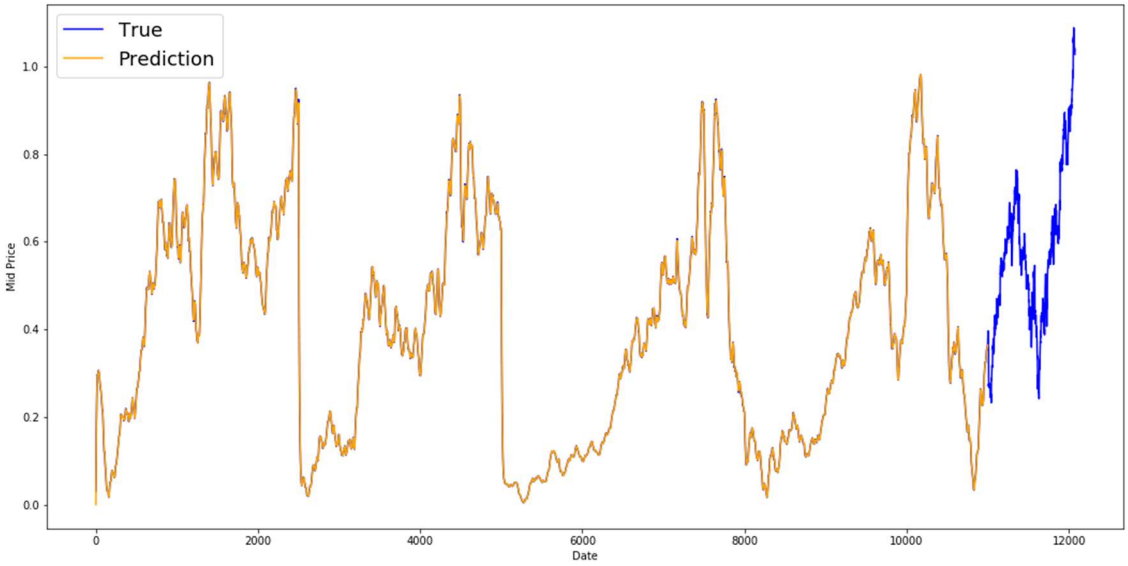
```
window_size = 100
```

```
いいえ train_data.サイズ
```

```
run_avg_predictions = []
```

```
run_avg_x = []
```

```
mse_errors = []

running_mean = 0.0
run_avg_predictions.append(running_mean)

減衰 = 0.5

範囲内の pred_idx( , 1,N):1

running_mean = running_mean*減衰 +(1.0-decay)*train_data[pred_idx-1]]
    run_avg_predictions.append(running_mean)
    mse_errors.append(run_avg_predictions[-1]-train_data[pred_idx])**2)
    run_avg_x.append(日付)

印刷('EMA 平均の MSE エラー: %.5f'%(0.5*np.mean(mse_errors)) )%(
EMA 平均の MSE エラー: 0.00003

plt.figure(figsize = (18,9))
plt.plot(範囲(df.shape[0])、all_mid_data、0 色='b'、ラベル='真')
plt.plot(範囲)(0,N),run_avg_predictions,色=オレンジ,ラベル='予測')0
#plt.xticks(範囲(0,df.shape[0],50)、df[日付].loc[::50]、回転=45)
plt.xlabel('日付')
plt.ylabel('ミッドプライス')
凡例（フォントサイズ=18)
plt.show()
```



指数移動平均が良い場合，なぜあなたはより良いモデルが必要なのですか?

それは真の分布に続く(そして非常に低い MSE によって正当化される)完璧な線に合うことがわかります。実質的に言えば、次の日の株式市場価値だけでは多くを行うことはできません。個人的に私が望むのは、次の日の正確な株式市場価格ではありませんが、*株式市場の価格は次の 30 日間で上がったり下がったりします*。これを試してみて、EMA メソッドの機能を公開します。

次に、ウィンドウで予測を行います（次の日ではなく、次の 2 日間のウィンドウを予測するとします)。その後、あなたは間違った EMA が行くことができる方法を実現します。例を次に示します。

未来への一歩以上を予測する

具体的なものにするには、xt=0.4xt=0.4、EMA =0.5EMA=0.5、γ =0.5 EMA=0.5 γ =0.5 と γ =0.5 仮定しましょう。

- 次の式で出力を取得するとします。
  - $X_{t+1} = EMA_t = \gamma \times EMA_{t-1} + (1 - \gamma)X_t$
  - したがって、$x_{t+1} = 0.5 \times 0.5 + (1-0.5) \times 0.4 = 0.45$
  - したがって、$X_{t+1} = EMA_t = 0.45$
- したがって、次の予測 X_{t+2} は、
  - $X_{t+2} = \gamma \times EMA_t + (1-\gamma)X_{t+1}$
  - $X_{t+2} = \gamma \times EMA\_t + (1-\gamma)\, EMA_t = EMA_t$
  - または、この例では、$X_{t+2} = X_{t+1} = 0.45$

したがって、将来の予測手順の数に関係なく、将来のすべての予測ステップで同じ答えを得続けることができます。

有益な情報を出力するソリューションの 1 つは、**モメンタムベースのアルゴリズム を見る方法です**。過去の最近の値が上がっていたか下がっているか(正確な値ではない)かに基づいて予測を行います。例えば、価格が過去数日間下がっている場合、翌日の価格は下がる可能性が高いと言うでしょう。ただし、より複雑なモデルである LSTM モデルを使用します。

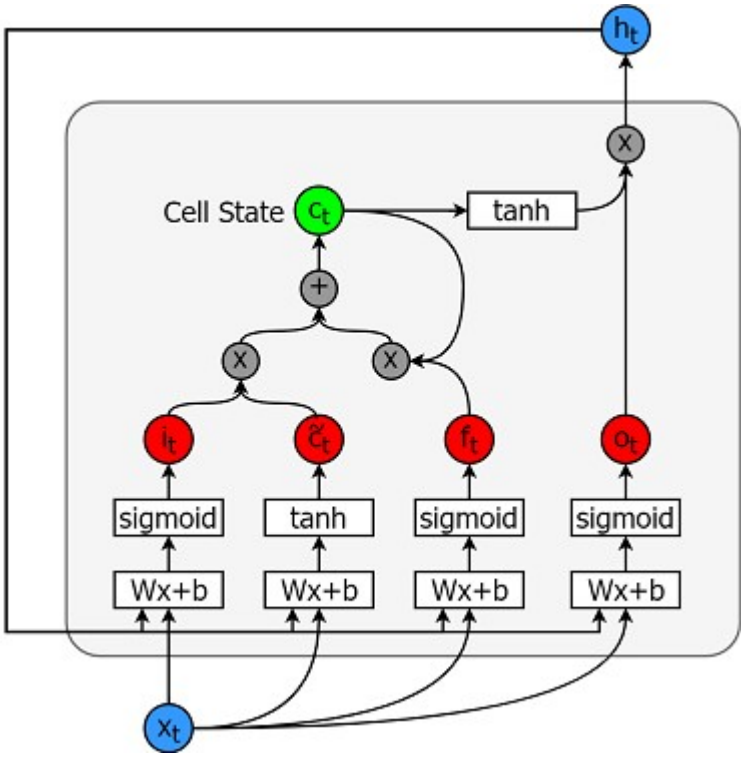
これらのモデルは、時系列データのモデリングが非常に得意であるため、嵐による時系列予測の領域を取りました。実際に、悪用できるデータに隠されたパターンがあるかどうかがわかります。

LSTMs の紹介:株式移動予測を未来に向けて進める

長期記憶モデルは非常に強力な時系列モデルです。彼らは将来のステップの任意の数を予測することができます。LSTM モジュール(またはセル)には 5 つの必須コンポーネントがあり、長期的なデータと短期データの両方をモデル化することができます。

- セル状態（$c_{t)}$- これは、短期記憶と長期記憶の両方を格納するセルの内部メモリを表します。
- 非表示状態（$h_{t)}$- これは、w.r.t. 現在の入力、以前の非表示状態、および将来の株式市場の価格を予測するために使用する現在のセル入力を計算した出力状態情報です。さらに、非表示状態は、短いメモリまたは長期のメモリ、またはセル状態に格納されているメモリの両方の種類のみを再設定して、次の予測を行うことを決定できます。
- 入力ゲート（$i_{t)}$- 現在の入力からセル状態への情報のフローを決定します。
- ゲートを忘れる（$f_{t)}$- 現在の入力から、前のセルの状態から現在のセル状態にフローする量を決定します。
- 出力ゲート（o_t) - 現在のセル状態からの情報が隠された状態に流れる量を決定し、必要に応じて LSTM が長期記憶または短期記憶と長期記憶のみを選択できるようにします。

セルは下に示します。



そして、これらの各々の実体を計算するための方程式は以下の通りです。

- $i_t = \sigma(W_{ix}X_t + W_{ih}h_{t-1} + b_i)$
- c~_t= σ (W_cxx_t+W_chht-1+bc)c~_t= σ (W_cxx_t +W_chht-1+bc)
- f_t= σ (W_fxx_t+W_fhht-1+bf)f_t= σ (W_fxx_t +W_fhht-1+bf)
- ct=f_tc_t-1+itc~tct=f_tc_t-1+itc~t
- o_t= σ (W_oxx_t+W_ああ t-1+bo)o_t= σ (W_oxx_t+W_ohht−1+bo)
- ht=ot ttanh(ct)ht=オタン(ct)

LSTM に関するより良い（より技術的な）理解のために、 この記事を参照してください。

TensorFlow は、時系列モデルを実装するための優れたサブ API (RNN API と呼ばれる）を提供します。実装に使用します。

データ ジェネレーター

まず、モデルをトレーニングするためのデータ ジェネレーターを実装します。このデータ ジェネレーターには、`.unroll_batches(.)` というメソッドがあります。これは、*num_unrollings* 連続して取得した入力データの num_unrollings バッチのセット `batch_size` を出力します。次に、入力データの各バッチに対応するデータの出力バッチが含まれます。

例えば、`num_unrollings=3` および `batch_size=4` の場合、一連の展開されていないバッチは次のようになります。

- 入力データ:,x12,31],[xx [xx0,,x10,x20,x30] [xx1,,x1,1,x 2 1,x21,x2,x 2, x 2,x2,x32]2][x0,x10,x20,x30],[x1,x11,x21,x31],x2,x12x2222

- 出力データ:x13,x12,[xxx31,,x1,1,x2,1,x31] , x2,,x2,x22, x3 2,x2 3, x233,x2 3,x33]3]3,[x1,x11,x21,x31],[x2,x12,x22,x32],x3,x133333

データ増強

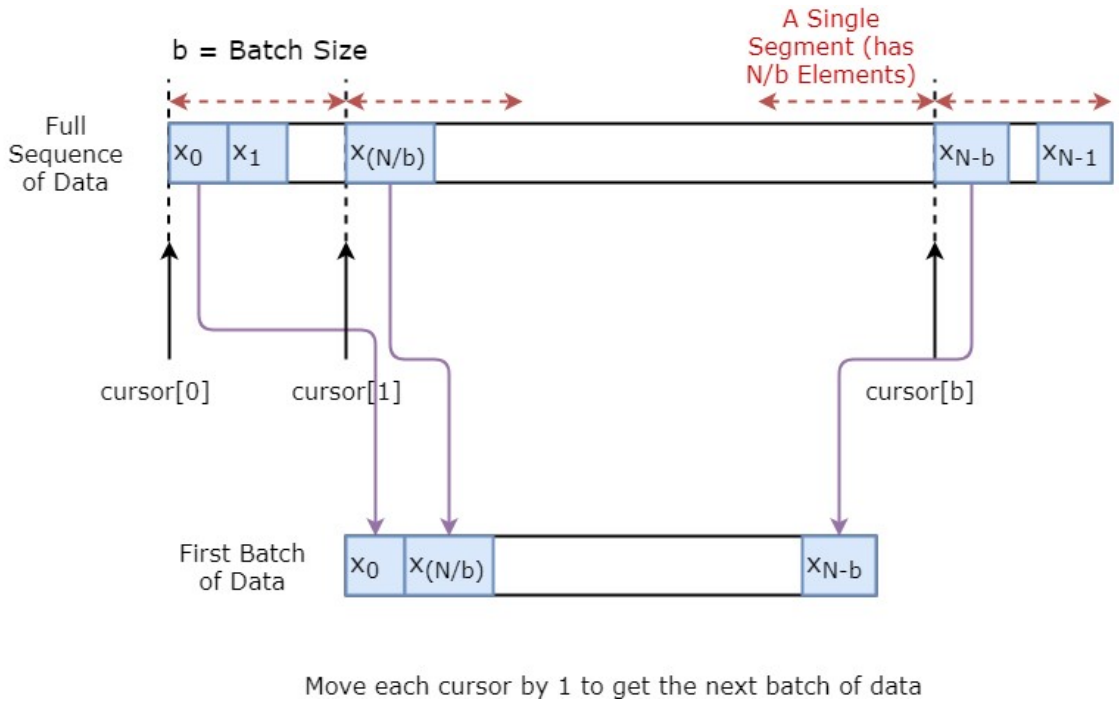
また、モデルを堅牢にするために、x_tx_t の出力を常に x__t+1 x_tt+1 にしません。むしろ、セット x NN_t+1、x_t+2,...,xtt+Nx_t+1,x_t +2,...,+N が小さいウィンドウ サイズの場合、出力をランダムにサンプリングします。

ここでは、次の仮定を行います。

- x_t+1,x_t+2,...,xt+Nx_t +1,x_t+2,...,xt+N は互いにあまり遠くない t

私は個人的にこれは株式移動予測のための合理的な仮定だと思います。

以下に、データのバッチを視覚的に作成する方法を示します。




```
クラス データジェネレーターセク(オブジェクト):

デフ __init____init__(自己、価格、batch_size、num_unroll):
        self._prices = 価格
        self._prices_length = len(self._prices) - num_unroll
        self._batch_size = batch_size
        self._num_unroll = num_unroll
        self._segments = self._prices_length //self._batch_size
```

self._cursor = [オフセット * 範囲内のオフセットの self._segments(self._batch_size)]

    デフ next_batch(自己):

        batch_data = np.zeros((self._batch_size)、dtype=np.float32)

        batch_labels = np.zeros(self._batch_size)、dtype=np.float32)

範囲(self._batch_size) の b の場合:

self._cursor[b]+1>=self._prices_length の場合:

                #self_カーソル[b] = b * self._segments

                self._cursor[b] = np.random.randint(0,(b+1)*self._segments)

            batch_data[b] = self._prices[self._cursor[b]]

batch_labels[b]= self._prices[self._cursor[b]+np.random.randint(0,0,5))]

            self._cursor[b] = (self._cursor[b]+1)%self._prices_length

batch_data,batch_labels を返す

    デフ unroll_batches(自己):

        unroll_data,unroll_labels = [],[]

        init_data、init_label = なし、なし

範囲 ui 内の ui(self._num_unroll):

            データ、ラベル = self.next_batch()

            unroll_data.append(データ)

            unroll_labels.append(ラベル)

unroll_data を返す, unroll_labels

```
    デフ reset_indices(自己):
範囲(self._batch_size) の b の場合:
self._cursor[b] = np.random.randint(0,min((b+1)*self._segments,self._prices_length-1))



dg = データジェネレーターSeq(train_data,5,5)
u_data、u_labels = dg.unroll_batches()

ui の場合 ,(dat,lbl) を 列挙(zip(u_data,u_labels)で指定します。
    印刷（'¥n¥n アンロール インデックス %d'%ui)
    dat_ind =
    lbl_ind = lbl
印刷:'
    印刷する(¥n¥t 出力:'、Lbl)
アンロールインデックス 0
    入力: [0.03143791 0.6904868 0.82829314 0.32585657 0.11600105]

    出力: [0.08698314 0.68685144 0.8329321 0.33355275 0.11785509]


アンロールインデックス 1
    入力: [0.06067836 0.6890754 0.8325337 0.32857886 0.11785509]

    出力: [0.15261841 0.68685144 0.8325337 0.33421066 0.12106793]


アンロールインデックス 2
    入力: [0.08698314 0.68685144 0.8329321 0.33078218 0.11946969]

    出力: [0.11098009 0.6848606 0.83387965 0.33421066 0.12106793]
```

```
アンロールインデックス 3
    入力: [0.11098009 0.6858036 0.83294916 0.33219692 0.12106793]

    出力: [0.132895 0.6836884 0.83294916 0.33219692 0.12288672]


アンロールインデックス 4
    入力: [0.132895 0.6848606 0.833369 0.33355275 0.12158521]

    出力: [0.15261841 0.6836884 0.83383167 0.33355275 0.12230608]
```

ハイパーパラメーターの定義

このセクションでは、いくつかのハイパーパラメーターを定義します。D は入力の次元です。前の株価を入力として受け取り、次の株価を予測すると、1..

次に、num_unrollings、これは LSTM モデルの最適化に使用される時間によるバックプロパゲーション（BPTT）に関連するハイパーパラメーターです。これは、1 つの最適化ステップで考慮する連続タイム ステップの数を示します。これは、単一の時間ステップを見てモデルを最適化するのではなく、num_unrollings 時間ステップを見てネットワークを最適化すると考えることができます num_unrollings。大きいほど良いです。

その後、あなたは batch_size を持っています.バッチ サイズは、1 つのタイム ステップで考慮するデータ サンプルの数です。

次に、各セル内の非表示のニューロンの数を表す num_nodes を定義します。この例では、LSTM の 3 つの層があることがわかります。

```
D = 1 # データの次元です。あなたのデータは 1-D なので、これは 1 になります
num_unrollings = 50 # 将来を調べている時間ステップの数。
batch_size = 500 # バッチ内のサンプル数
num_nodes = [200,200,150]# 使用している深い LSTM スタックの各層の隠しノードの数
n_layers = len(num_nodes) # レイヤー数
ドロップアウト = 0.2 # ドロップアウト量

tf.reset_default_graph() # これを複数回実行する場合に重要です
```

入力と出力の定義

次に、トレーニング入力とラベルのプレースホルダを定義します。入力プレースホルダのリストがあり、各プレースホルダにはデータのバッチが 1 つ含まれているので、これは非常に簡単です。リストには num_unrollings のプレースホルダがあり、1 回の最適化ステップで一度に使用されます。

```
# 入力データ。

train_inputs、train_outputs = [],[]


# 時間ステップごとにプレースホルダを定義する入力を時間をかけて展開します。

範囲内の ui(num_unrollings):

    train_inputs.append(tf.float32,shape=[batch_size,D],名前='%d'%ui)を train_inputs_します)

train_outputs.append(tf.float32、図形=[batch_size,1],名前 1= 'train_outputs_%d'%ui))
```

LSTM および回帰レイヤーのパラメーターの定義

LSTM の 3 つの層と、最後 and の b 長期メモリ セルの出力を取得し、次の時間ステップの予測を出力する、w と b で示される線形回帰レイヤーがあります。テンソルフローのマルチ RNN セルを使用して、作成した 3 つの LSTMCell オブジェクトをカプセル化できます。また、パフォーマンスを向上させ、オーバーフィットを減らすため、ドロップアウトの LSTM セルを実装することもできます。

```
lstm_cells = [

    tf.contrib.rnn.LSTM セル(num_units=num_nodes[li]、

                            state_is_tuple =True、

                            初期化子= tf.contrib.layers.xavier_initializer()

                           )

範囲の li(n_layers) ]


drop_lstm_cells = [tf.contrib.rnn.ドロップアウトラッパー(

    lstm, input_keep_prob =1.0,output_keep_prob=1.0-ドロップアウト, state_keep_prob =1.0-ドロップアウト

) の lstm の lstm_cells] ]

drop_multi_cell = tf.contrib.rnn.MultiRNNCell(drop_lstm_cells)

multi_cell = tf.contrib.rnn.MultiRNNCell(lstm_cells)


w = tf.get_variable('w',shape=[num_nodes[-1] , 1], 初期化子=tf.contrib.layers.xavier_initializer())

b = tf.get_variable('b',初期化子=tf.random_uniform(11],-0.1, 0.1,0.1) ) ],
```

LSTM 出力を計算し、回帰レイヤーに供給して最終的な予測を得る

このセクションでは、まず、c and h セルの状態と長期メモリセルの非表示状態を保持する TensorFlow 変数（c および h）を作成します。次に、train_inputs のリストを変換 train_inputs して[num_unrollings、batch_size、D]のシェイプを持ち、tf.nn.dynamic_rnn 関数で出力を計算するために必要です tf.nn.dynamic_rnn。次に、tf.nn.dynamic_rnn 関数を使用して LSTM 出力を計算 tf.nn.dynamic_rnn し、予測と真の株価の間の損失を num_unrolling テンソルのリストに戻します。

```
# LSTM の状態を維持するために、セル状態と非表示状態変数を作成します。
c, h - [],[]
initial_state = []
範囲(n_layers) の li の場合:
  c.append(tf.変数(tf.ゼロ([batch_size,num_nodes[li])、トレーニング可能=偽) )
  を追加します。変数(tf.ゼロ([batch_size,num_nodes[li])、トレーニング可能=偽) )
  initial_state.append (tf.contrib.rnn.LSTMStateTuple(c[li], h[li])

# 関数 dynamic_rnn は、出力が必要であるため、いくつかのテンソルトランスフを行います
# 特定の形式。続きを読む: https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn
all_inputs = tf.concat([tf.expand_dims(tf.expand_dims(t,0)for の t の train_inputs] ,軸 =0)

#all_outputs は[seq_length、batch_size、num_nodes] です
all_lstm_outputs、状態 = tf.nn.dynamic_rnn(
    drop_multi_cell、all_inputs、initial_state=タプル(initial_state))
    time_major = 真、dtype=tf.float32)

all_lstm_outputs = tf.reshape(all_lstm_outputs、[batch_size*num_unrollings,num_nodes[-1])-1]])

all_outputs = tf.nn.xw_plus_b(all_lstm_outputs,w,b)

split_outputs = tf.split(all_outputs,num_unrollings,軸=0))
```

損失計算とオプティマイザー

次に、損失を計算します。ただし、損失を計算する際には固有の特性があることに注意してください。予測のバッチと真の出力ごとに、平均二乗誤差を計算します。そして、あなたはこれらすべての平均平方損失を一緒に(平均では

ない)合計します。最後に、ニューラル ネットワークの最適化に使用するオプティマイザを定義します。この場合、非常に最近のパフォーマンスの高いオプティマイザである Adam を使用できます。

```
#あなたが計算するので、正確なフォームについて注意する必要がある損失を計算するとき
#同時にすべての展開されたステップの損失
したがって#、平均エラーまたは各バッチを取り、すべての展開されたステップの上にその合計を取得します

印刷('トレーニングロスの定義')
損失 = 0.0
tf.control_dependencies([tf.assign(c[li],state[li][0])0 を for 指定して範囲内の li(n_layers)+
[tf.assign(h[li], state[li][1])範囲 for 内の li の場合(n_layers)]):
範囲内の ui(num_unrollings):
    損失 += tf.reduce_mean (0.5*(split_outputs[ui]train_outputs[ui])**2)

印刷('学習速度減衰操作')
global_step = tf.変数(0, トレーニング可能=偽)
inc_gstep = tf.assign(global_step,global_step + 1))
tf_learning_rate = tf.placeholder(図形=なし、dtype=tf.float32)
tf_min_learning_rate = tf.placeholder(図形=なし、dtype=tf.float32)

learning_rate = tf.maximum(
    tf.train.exponential_decay(tf_learning_rate、global_step、decay_steps=1、decay_rate=0.5、階段=真)
    tf_min_learning_rate)

#オプティマイザー。
印刷('TF 最適化操作')
オプティマイザー = tf.train.アダムオプティマイザー(learning_rate)
グラデーション, v = zip(*optimizer.compute_gradients(損失))
グラデーション, _ = tf.clip_by_global_norm(グラデーション、 5.0)
オプティマイザ = optimizer.apply_gradients(
    ジップ(グラデーション、v))

印刷 ('¥tall 完了')
```

トレーニング損失の定義

学習速度減衰操作

TF 最適化操作

すべて完了

予測関連計算

ここでは、予測関連のテンソルフロー操作を定義します。まず、入力(sample_inputs)にフィードするためのプレースホルダを定義し、次にトレーニングステージと同様に、予測(sample_c と sample_h)の状態変数を定義 sample_c and sample_h します。最後に、tf.nn.dynamic_rnn 関数で予測を計算 tf.nn.dynamic_rnn し、回帰レイヤー(w と b)を通して出力を送信します。また、reset_sample_state セルの状態と非表示状態をリセットする reset_sample_state 操作も定義する必要があります。この操作は、予測のシーケンスを作成するたびに、開始時に実行する必要があります。

```
印刷('予測関連の TF 関数の定義')


sample_inputs = tf.placeholder(tf.float32,shape=[11,D])


# 予測段階の LSTM 状態の維持

sample_c、sample_h、initial_sample_state=[][][][

範囲(n_layers) の li の場合:

  sample_c.append(tf.変数(tf.ゼロ(1,num_nodes[li]), トレーニング可能=偽) )

  sample_h.append(tf.変数(tf.ゼロ(1,num_nodes[li]), トレーニング可能=偽) )

  initial_sample_state.append(tf.contrib.rnn.LSTMStateTuple(sample_c[li],sample_h[li])


reset_sample_states = 範囲内 range(n_layers)],の forli の tf.group(*[tf.assign(sample_c[sample_c[sample_c[],tf.zeros([1,num_nodes[n_layers]]))1)

*[tf.assign(sample_h[li],tf.ゼロ li infor([1,num_nodes[li])]n_layers) 1


sample_outputs、 sample_state = tf.nn.dynamic_rnn(multi_cell、tf.expand_dims(sample_inputs,0)

                                  initial_state=タプル(initial_sample_state)

                                  time_major = True、

                                  d タイプ=tf.フロート 32)


for li in 範囲(n_layers)]の tf.control_dependencies([tf.assign(sample_c[li]、sample_state[li][0])0

[tf.assign(sample_h[li],sample_state[li][1]n_layers)1 for  in
```

```
sample_prediction = tf.nn.xw_plus_b (tf.reshape(sample_outputs、1、-1-1)、w、 b)


印刷 ('¥tall 完了')

予測関連の TF 関数の定義

    すべて完了
```

LSTM の実行

ここでは、いくつかのエポックの株価の動きを訓練し、予測を予測し、時間の経過とともに予測が良くなるか悪化するかを確認します。次の手順に従います。

- モデルを評価する時系列の開始点 (test_points_seq) のテスト セットを定義します。
- エポックごとに
  - トレーニングデータの全シーケンス長
  - 一連の num_unrollings バッチのロールをアンロールする
  - 展開されたバッチを使用してニューラル ネットワークをトレーニングする
  - 平均トレーニング損失の計算
  - テスト セットの開始点ごとに
  - テスト ポイントの前に見つかった以前の num_unrollings データ ポイントを反復処理して、LSTM 状態を更新します。
  - 前の予測を現在の入力として使用して、n_predict_once ステップの予測を継続的に行う n_predict_once
  - 予測された n_predict_once ポイントと、それらのタイムスタンプで真の株価との間の MSE 損失を計算します

```
エポック = 30

valid_summary = 1 # テスト予測を行う間隔


n_predict_once = 50 # 連続して予測するステップ数


train_seq_length = train_data.size # トレーニング データの全長


train_mse_ot = [] # 列車の損失を蓄積する

test_mse_ot = [] # テスト損失を累積する

predictions_over_time = [] # 予測を累積する
```

セッション ＝tf.インタラクティブセッション()

tf.global_variables_initializer()を実行します。

#学習率を低下させるに使用

loss_nondecrease_count = 0

loss_nondecrease_threshold = 2 # この多くのステップでテストエラーが増加していない場合は、学習率を下げる

印刷（'初期化'）

average_loss = 0

データ ジェネレーターの定義

data_gen = データジェネレーターセク(train_data,batch_size,num_unrollings)

x_axis_seq = []

#からあなたのテスト予測を開始ポイント

test_points_seq = np.arange(11000,12000,50).tolist()

範囲(エポック)の ep の場合:

=

==========

==========

=

範囲 step 内のステップ(train_seq_length//batch_size):

u_data、u_labels = data_gen.unroll_batches()

feed_dict = {}

ui の場合 ,(dat,lbl) を 列挙(zip(u_data,u_labels)で指定します。

```
feed_dict[train_inputs[タマネギ]=その.reshape(-1、1)-1,1)

                feed_dict[train_outputs[ui]= lbl.reshape(-1,1)


          feed_dict.update({tf_learning_rate: 0.0001,tf_min_learning_rate:0.000001})


          _, l = session.run([オプティマイザー、損失]、feed_dict=feed_dict)


          average_loss


    # =
==========================================================================================

if (ep+1) % valid_summary==   0:


          average_loss = average_loss/(valid_summary*(train_seq_length//batch_size)


          #平均損失

if (ep+1)%valid_summary==0:

印刷('ステップ  %d  での平均損失: %f'  % (ep+1, average_loss)


          train_mse_ot.append(average_loss)


average_loss = 0   #  リセット損失


          predictions_seq = []


          mse_test_loss_seq = []


          # ====================  状態の更新とプレディシトンの作成
==========================================================================================

====== = =

test_points_seq の w_i:


          mse_test_loss = 0.0
```

```
            our_predictions = []

if (ep+1)-valid_summary==0:
                # 最初の検証エポックで x_axis 値のみを計算する
                x_axis=[]

            #株価の最近の過去の行動のフィード
            #その時点から予測を行う
 in 範囲内の tr_i(w_i-num_unrollings+ 11,w_i-1):):
                current_price = all_mid_data[tr_i]
                feed_dict[sample_inputs] = np.array(current_price) の形状変更(1,1)
                _ = セッション.run(sample_prediction,feed_dict=feed_dict)

            feed_dict = {}

current_price = all_mid_data[w_i-1]]

            feed_dict[sample_inputs] = np.array(current_price) の形状変更(1,1)

            #この多くのステップの予測を行います
            #各予測は、現在の入力として以前のプレジトンを使用しています
 in 範囲内の pred_i(n_predict_once):

                pred = session.run(sample_prediction,feed_dict=feed_dict)

                our_predictions.append(np.アススカラ(プレット))

                feed_dict[sample_inputs] = np.asarray(pred).reshape(-1,1)

if (ep+1)-valid_summary==0:
                # 最初の検証エポックで x_axis 値のみを計算する
                x_axis.append(w_i+pred_i)
```

```
                mse_test_loss += 0.5*(プレ all_mid_data[w_i+pred_i])**2

            セッション実行(reset_sample_states)

            predictions_seq.append(our_predictions 配列))

            n_predict_once mse_test_loss /=

            mse_test_loss_seq.append(mse_test_loss)

if (ep+1)-valid_summary==0:

                x_axis_seq.append(x_axis)

        current_test_mse = np.mean(mse_test_loss_seq)

        #学習速度減衰ロジック

len(test_mse_ot)>0 と current_test_mse > 分 (test_mse_ot):

                loss_nondecrease_count += 1

        それ以外:

                loss_nondecrease_count = 0

loss_nondecrease_count > loss_nondecrease_threshold の場合：

                    実行(inc_gstep)

                loss_nondecrease_count = 0

                印刷 ('¥t 学習率を 0.5 ずつ減らす')

        test_mse_ot.append(current_test_mse)

        印刷 ('¥tTest MSE: %.5f'%np.mean(mse_test_loss_seq)

        predictions_over_time.append(predictions_seq)

        印刷 ('¥t 完成予測')

初期化

ステップ 1 での平均損失:1.703350
```

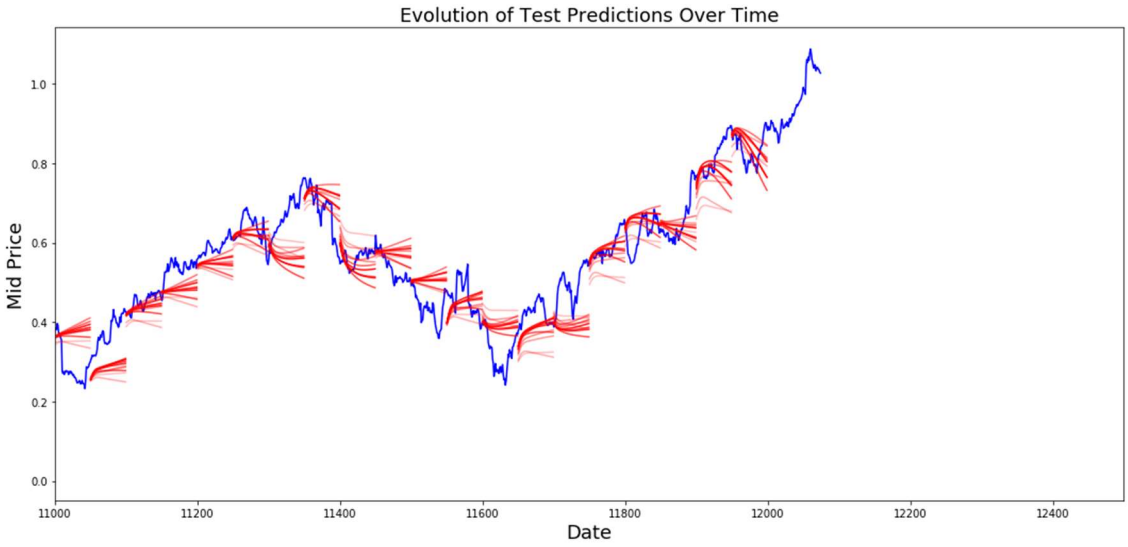
```
    テスト  MSE: 0.00318
    完成した予測
...
...
...
ステップ 30 での平均損失:0.033753
    テスト  MSE: 0.00243
    完成した予測
```

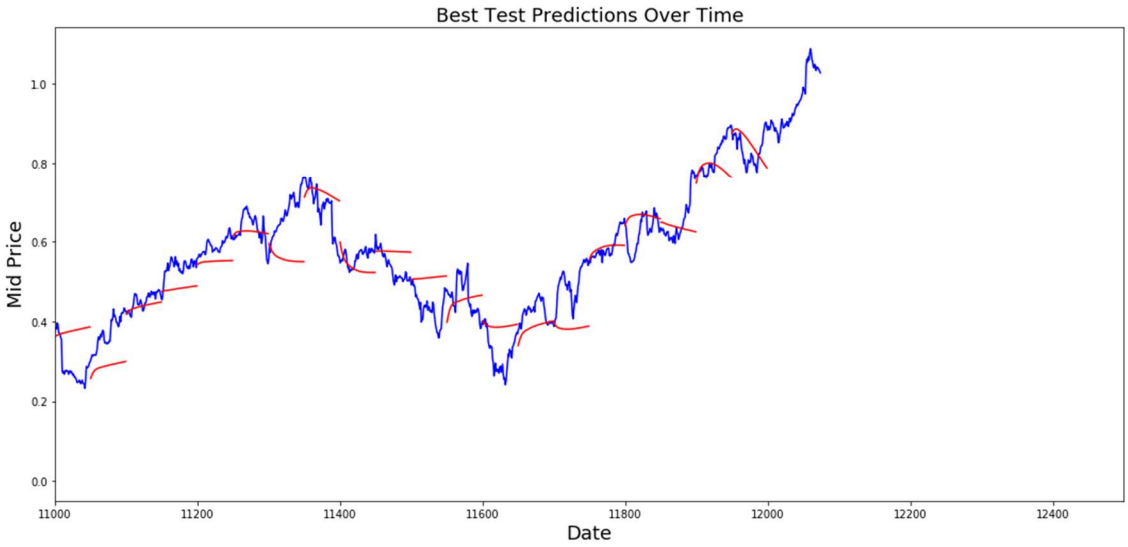
予測の視覚化

トレーニングの量で MSE の損失がどのように減少しているかを見ることができます。これは、モデルが何か役に立つことを学んでいるという良い兆候です。調査結果を定量化するために、ネットワークの MSE 損失を、標準平均（0.004）を実行するときに得た MSE 損失と比較できます。LSTM は標準平均よりも優れていることがわかります。そして、あなたは標準平均化(完璧ではないが)合理的に真の株価の動きに従っていることを知っています。

```
best_prediction_epoch = 28 # プロットコードの実行時に最良の結果が得られるエポックに置き換えます

plt.figure(フィグサイズ =(18,,18))
plt.サブプロット(2,1,1)
plt.plot(範囲(df.shape[0]),all_mid_data,色='b'))

#予測が時間の経過とともにどのように変化するかをプロットする
#高アルファで低いアルファと新しい予測で古い予測をプロット
start_alpha = 0.25
アルファ  = np.arange(start_alpha,1.1,(1.0-start_alpha)/len(predictions_over_time[::3]),(
列挙 p_i の場合(predictions_over_time[::3]):3]):
xval,yval の zip(x_axis_seq,p):
            plt.plot(xval,yval,カラー='r',アルファ=アルファ[p_i])

plt.title(「時間経過によるテスト予測の進化」、フォントサイズ=18))
plt.xlabel('日付'、フォントサイズ=18))
plt.ylabel('ミッドプライス'、フォントサイズ=18))
plt.xlim(11000,12500)
```

```
plt.サブプロット(2,1,2)


#あなたが得た最高のテスト予測を予測する

plt.plot(範囲(df.shape[0]),all_mid_data,色='b'))

xval,yval の場合(x_axis_seq,predictions_over_time[best_prediction_epoch]):

plt.plot(xval,yval,色='r'))


plt.title('時間の経過に対する最良のテスト予測'、フォントサイズ=18))

plt.xlabel('日付'、フォントサイズ=18))

plt.ylabel('ミッドプライス'、フォントサイズ=18))

plt.xlim(11000,12500)

plt.show()
```





完璧ではありませんが、LSTM はほとんどの場合、株価の行動を正しく予測できるようです。予測は、およそ 0 と 1.0 の範囲で行っています（つまり、実際の株価ではありません)。これは、価格自体ではなく、株価の動きを予測しているので、大丈夫です。

最終注釈

私はあなたがこのチュートリアルが便利だと思っていることを願っています。私はこれが私にとってやりがいのある経験だったことを言及する必要があります。このチュートリアルでは、株価の動きを正しく予測できるモデルをデバイス化するのがいかに難しいかを学びました。株価をモデル化する必要がある理由の動機から始めました。その後、データをダウンロードするための説明とコードが続きました。次に、予測を将来の 1 段階で行えるようにする 2 つの平均化手法を見ました。次に、これらのメソッドは、将来の複数のステップを予測する必要がある場合に無駄であることを確認しました。その後、LSTM を使用して将来の予測を行う方法について説明しました。最後に、結果を視覚化し、あなたのモデル(完璧ではないが)が株価の動きを正しく予測するのに非常に優っていることを見ました。

ディープラーニングについて詳しく知りたい場合は、Python のディープラーニングコースをご覧ください。 それは基本だけでなく、Keras で自分でニューラルネットワークを構築する方法をカバーしています。これは、このチュートリアルで使用される TensorFlow とは異なるパッケージですが、アイデアは同じです。

ここでは、このチュートリアルのいくつかのテイクアウトを述べています。

1. 株価・動き予測は非常に困難な作業です。個人的には、そこに出る株式予測モデルは当たり前だと思うべきではないと思います。しかし、モデルはほとんどの場合、株価の動きを正しく予測できるかもしれませんが、必ずしもそうではありません。
2. 真の株価と完全に重なる予測曲線を示す記事にだまされてはいけません。これは簡単な平均化技術で複製することができ、実際には役に立ちません。より賢明なことは、株価の動きを予測することです。
3. モデルのハイパーパラメーターは、得られる結果に非常に敏感です。そのため、ハイパーパラメーターの最適化テクニック（グリッド検索/ランダム検索など）をハイパーパラメーターで実行することが非常に良いことです。以下私は最も重要なハイパーパラメータのいくつかを挙げました
   - オプティマイザの学習率
   - 各レイヤーのレイヤー数と隠れユニット数
   - オプティマイザー。私はアダムが最高のパフォーマンスを見つけた
   - モデルの型。ピープホールと評価性能の違いを持つ GRU / 標準 LSTM / LSTM を試すことができます
4. このチュートリアルでは、(データの小さなサイズのために)欠陥のある何かをしました!それはあなたが学習率を減衰させるためにテスト損失を使用しています。これにより、テストセットに関する情報がトレーニン

グ手順に間接的に漏洩します。これを処理する優れた方法は、検証セットのパフォーマンスに関して(テストセットとは別に)個別の検証セットと減衰学習率を持つ方法です。

私と連絡を取りたい場合は、thushv@gmail.com で私に電子メールをドロップするか、LinkedIn で私と接続することができます。

参照

私は、株式予測に LSTM を使用する方法についての理解を得るために、このリポジトリに言及しました。しかし、詳細はリファレンスで見つかった実装とは大きく異なる場合があります。