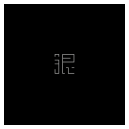
プロフィール



ID: **Gin04**
Twitter: **Gin04tw**
GitHub: **Gin04gh**

データ分析業務、予測モデル開発などを担当しています。
勉強してきた技術などを適当に書いていければと思います。
Python、R、SAS、JavaScript、PHP、Rubyらへんが得意です。

最近の投稿

人気の投稿

畳み込みニューラルネットワークで為替予測してみたかった

2017-10-13
Python, 深層学習

タイトルの通り、為替データを畳み込みニューラルネットワーク（CNN）で読み込んで予測させるまでを、Chainerで実装してみました。

今回は順を追って、データの取得と加工、モデルの構築、学習と予測をしていきます。

まずは為替のデータの取得です。

Pythonのライブラリ `pandas_datareader` を使って、日付ごとのドル円価格のデータを取得しました。

期限は適当に区切って、7年分を取得してきています。

```
import numpy as np
import pandas as pd
from pandas_datareader import data, wb

start = "2010-08-01"
end = "2017-08-01"
usdjpy = data.DataReader("DEXJPUS", "fred", start, end) # ドル円レート from セントルイス連邦準備銀行
display(usdjpy.head())

```

	DEXJPUS
DATE	
2010-08-02	86.42
2010-08-03	85.88
2010-08-04	86.26
2010-08-05	85.83
2010-08-06	85.25

いったん様子を確認するため、プロットしてみます。

```
import matplotlib
import matplotlib.pylab as plt
matplotlib.style.use("ggplot")

# 原系列プロット
usdjpy_tmp = usdjpy.dropna()
usdjpy_tmp.plot(color="cyan", figsize=(15,7))
plt.show()

```

さて、この時系列のデータから、ある程度、値の動きを読み込ませた後にその次の値は上がるか・下がるかの予測をできるかどうか、モデルを構築して試してみます。

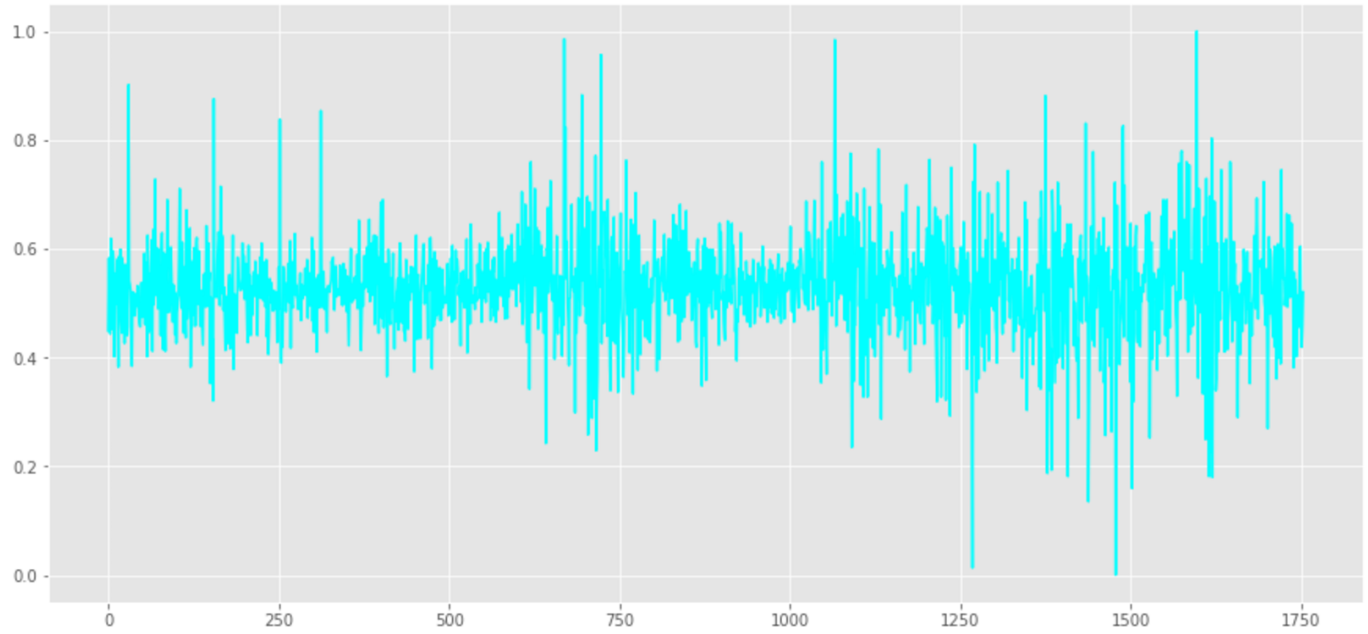
方法としては、ある程度の時系列の長さを決め、その間の値の上がり具合・下がり具合を数値化したものをCNNで読み込んで、次は「上がる」か「下がる」の分類モデルを構築することにします。

まずは原系列から差分系列に変換し、上下の具合を数値化するため、[0,1]区間に正規化します。

```
from sklearn.preprocessing import MinMaxScaler

# データ加工
scaler = MinMaxScaler(feature_range=(0,1))
usdjpy_tmp = usdjpy.dropna()
usdjpy_tmp = usdjpy_tmp.diff().dropna() # 差分（変動）の系列にする
usdjpy_tmp = np.array(usdjpy_tmp["DEXJPUS"].values).reshape(-1,1) # numpy配列にする
usdjpy_tmp = scaler.fit_transform(usdjpy_tmp) # 正規化
plt.figure(figsize=(15,7))
plt.plot(usdjpy_tmp, color="cyan")
plt.show()

```



次に、CNNで読み込める教師データの形にします。

系列データから決めた時系列の長さ分だけ取得してきて、入力を上下の具合を１次元の画像と見立てたもの、出力を1（上がる）か0（下がる）に変換していきます。

```
SEQ_SIZE = 50

dataset = []
for i in range(len(usdjpy_tmp)-SEQ_SIZE): # SEQ_SIZE分の系列をとり、その次の値が上がっているか下がっているかのラベルにする
    x = usdjpy_tmp[i:i+SEQ_SIZE]
    t_tmp = usdjpy_tmp[i+SEQ_SIZE]
    t = 1 # 上がっている
    if x[-1] &gt; t_tmp: # 下がっている
        t = 0
    x = np.array(x, dtype="float32").reshape(1, 1,SEQ_SIZE) # 畳み込みニューラルネットワークにするので、時系列は１次元の画像と見立てる（in_channel, height_size, width_siz
    t = np.array(t, dtype="int32")
    dataset.append((x,t))
N = len(dataset)

print("教師データ件数: ", N)

# 一部確認してみる
for i in range(5):
    print(dataset[i][0][0])
    plt.figure(figsize=(30,140))
    plt.imshow(dataset[i][0][0], cmap="gray")
    plt.show()

```

```
教師データ件数: 1704
[[ 0.44927537  0.5826087   0.46521738  0.44347826  0.61884058  0.52753621
   0.45217392  0.59565216  0.58115941  0.40144926  0.5536232   0.49855071
   0.52173913  0.57971013  0.45942029  0.38260868  0.58550727  0.53478259
   0.59855074  0.46521738  0.44492754  0.5826087   0.47971013  0.57101446
   0.42608696  0.54202896  0.5246377   0.57101446  0.44057971  0.45797101
   0.90144926  0.54057974  0.53478259  0.52318841  0.47246376  0.38405797
   0.52028984  0.51594204  0.51739132  0.47681159  0.49710146  0.51594204
   0.49420291  0.53188407  0.5043478   0.47391304  0.46231884  0.45652175
   0.51014495  0.53768116]]
```

```
[[ 0.5826087   0.46521738  0.44347826  0.61884058  0.52753621  0.45217392
   0.59565216  0.58115941  0.40144926  0.5536232   0.49855071  0.52173913
   0.57971013  0.45942029  0.38260868  0.58550727  0.53478259  0.59855074
   0.46521738  0.44492754  0.5826087   0.47971013  0.57101446  0.42608696
   0.54202896  0.5246377   0.57101446  0.44057971  0.45797101  0.90144926
   0.54057974  0.53478259  0.52318841  0.47246376  0.38405797  0.52028984
   0.51594204  0.51739132  0.47681159  0.49710146  0.51594204  0.49420291
   0.53188407  0.5043478   0.47391304  0.46231884  0.45652175  0.51014495
   0.53768116  0.47391304]]
```

```
[[ 0.46521738  0.44347826  0.61884058  0.52753621  0.45217392  0.59565216
   0.58115941  0.40144926  0.5536232   0.49855071  0.52173913  0.57971013
   0.45942029  0.38260868  0.58550727  0.53478259  0.59855074  0.46521738
   0.44492754  0.5826087   0.47971013  0.57101446  0.42608696  0.54202896
   0.5246377   0.57101446  0.44057971  0.45797101  0.90144926  0.54057974
   0.53478259  0.52318841  0.47246376  0.38405797  0.52028984  0.51594204
   0.51739132  0.47681159  0.49710146  0.51594204  0.49420291  0.53188407
   0.5043478   0.47391304  0.46231884  0.45652175  0.51014495  0.53768116
   0.47391304  0.51014495]]
```

```
[[ 0.44347826  0.61884058  0.52753621  0.45217392  0.59565216  0.58115941
   0.40144926  0.5536232   0.49855071  0.52173913  0.57971013  0.45942029
   0.38260868  0.58550727  0.53478259  0.59855074  0.46521738  0.44492754
   0.5826087   0.47971013  0.57101446  0.42608696  0.54202896  0.5246377
   0.57101446  0.44057971  0.45797101  0.90144926  0.54057974  0.53478259
   0.52318841  0.47246376  0.38405797  0.52028984  0.51594204  0.51739132
   0.47681159  0.49710146  0.51594204  0.49420291  0.53188407  0.5043478
   0.47391304  0.46231884  0.45652175  0.51014495  0.53768116  0.47391304
   0.51014495  0.5043478 ]]
```

一部データを確認してみたものが上記となります。

上記のように、値がどう動いてきたかを色の濃淡の画像で表す感じになります。

途中とても白い部分がありますが、ここが原系列の中でもかなり大きく値が変動した部分となっているようです。

実際の値を確認してみると、 0.90144926 がその箇所のようです。

これを分類するCNNをChainerで組んでいきます。

入力する画像（時系列）は縦方向の次元が 1 次元となっていますので、フィルターの縦サイズは 1 とし、横サイズは適当に決めることにします。

出力層の次元数は、1（上がる）か0（下がる）なので2です。

```
import chainer
from chainer import Chain, optimizers, training
from chainer.training import extensions
import chainer.functions as F
import chainer.links as L

# CNNクラスの定義
class CNN(Chain):
    def __init__(self):
        # クラスの初期化
        super(CNN, self).__init__(
            conv1 = L.Convolution2D(1, 20, (1,5)),
            conv2 = L.Convolution2D(20, 50, (1,5)),
            l1 = L.Linear(500, 100),
            l2 = L.Linear(100, 100),
            l3 = L.Linear(100, 2)
        )

    def __call__(self, x):
        # 順伝播の計算を行う関数
```

```
        # :param x: 入力値
        h = F.max_pooling_2d(F.relu(self.conv1(x)), 2)
        h = F.max_pooling_2d(F.relu(self.conv2(h)), 2)
        h = F.relu(self.l1(h))
        h = F.relu(self.l2(h))
        y = self.l3(h)
        return y

```

上記のように、畳み込み層2層＋全結合3層のモデルを用意しました。

これに先ほど作成した教師データから、学習用、テスト用に分割し、学習と予測精度の評価をしてみます。

学習の実装が下記になります。

```
EPOCH_NUM = 30
BATCH_SIZE = 200

# モデルの定義
model = L.Classifier(CNN())
optimizer = optimizers.Adam()
optimizer.setup(model)

# 学習開始
print("Train")
train, test = chainer.datasets.split_dataset_random(dataset, N-100)
train_iter = chainer.iterators.SerialIterator(train, BATCH_SIZE)
test_iter = chainer.iterators.SerialIterator(test, BATCH_SIZE, repeat=False, shuffle=False)
updater = training.StandardUpdater(train_iter, optimizer, device=-1)
trainer = training.Trainer(updater, (EPOCH_NUM, "epoch"), out="result")
trainer.extend(extensions.Evaluator(test_iter, model, device=-1))
trainer.extend(extensions.LogReport())
trainer.extend(extensions.PrintReport( ["epoch", "main/loss", "validation/main/loss", "main/accuracy", "validation/main/accuracy", "elapsed_time"])) # エポック
#trainer.extend(extensions.ProgressBar()) # プログレスバー出力
trainer.run()

```

```
Train
epoch       main/loss   validation/main/loss  main/accuracy  validation/main/accuracy  elapsed_time
1           0.6945      0.696726              0.512778       0.5                       0.368694
2           0.69371     0.693894              0.51125        0.5                       0.668306
3           0.693992    0.69406               0.501875       0.5                       0.960832
4           0.693212    0.693427              0.48125        0.5                       1.24069
5           0.692661    0.693507              0.50125        0.49                      1.59873
6           0.692001    0.693523              0.53625        0.52                      1.91426
7           0.69139     0.69376               0.533125       0.43                      2.23604
8           0.68936     0.693013              0.531875       0.47                      2.53556
9           0.686648    0.691831              0.55875        0.53                      2.86279
10          0.683469    0.691232              0.55625        0.53                      3.18392
11          0.681901    0.684926              0.5675         0.53                      3.49856
12          0.66974     0.684411              0.62375        0.57                      3.8214
13          0.658943    0.680541              0.619375       0.58                      4.15561
14          0.636122    0.67823               0.659375       0.55                      4.46374
15          0.619519    0.689253              0.66           0.62                      4.74584
16          0.600674    0.691189              0.665625       0.62                      5.05895
17          0.602458    0.671296              0.675625       0.64                      5.36039
18          0.565715    0.673815              0.71125        0.65                      5.64803
19          0.557535    0.673052              0.720625       0.65                      5.93639
20          0.565635    0.68234               0.700625       0.62                      6.30144
21          0.556392    0.672347              0.700625       0.62                      6.62216
22          0.566502    0.668243              0.7            0.65                      6.92982
23          0.539419    0.67359               0.7425         0.66                      7.22881
24          0.516406    0.665725              0.7425         0.67                      7.55725
25          0.520564    0.666941              0.7325         0.66                      7.85159
26          0.503979    0.683632              0.75125        0.63                      8.14055
27          0.506206    0.663624              0.74625        0.7                       8.43634
28          0.484892    0.655963              0.7625         0.69                      8.72336
29          0.477316    0.693221              0.77           0.63                      9.00791
30          0.479508    0.691833              0.765          0.67                      9.3656
```

テストデータに対して、7割近くまで当てられるようになりました。

~~個人的には、思っていた以上に当たっていて、びっくりしています笑~~

~~こうしてニューラルネットワークである程度予測できるということは、やはり何かしら規則的なものがあるのでしょうか。~~

~~素人からすると次が上がるか下がるかなんて全く予測できませんが、経験多く積んだ投資家や専門家から見ると、少なくともこの程度の精度以上には当てられるのかもしれません。~~

~~株価の世界などで俗に言われるテクニカル分析というものが、レートの形から将来の値動きを予測する手法と言われていますので、何かしら予測する方法があるのだろうと思います。~~

~~ところで、今回はCNNを使いましたが、こういった時系列データはむしろ再帰的ニューラルネットワーク（RNN）の方が合っているのではないかと思いますが、実はRNNでも試しており、同程度の予測率が出せることが確認できています。~~

~~ただしRNNですと、やはり計算量が大きくなりがちですので、より早いモデルとしてCNNを使った場合はどうなるのかの検証のため、今回はCNNを試してみたといったところです。~~

~~フィルターで読み取った上でマックスプーリングして、ある程度、事前の系列を集約してしまった上でも、先ほどのような規則的なものが現れると考えても良さそうです。~~

追記（2017-12-16）

記事を作成した時は「絶対何か間違いがあるだろうなー」とは思っていましたが、やはり正しくないことをしていたようです。

下記の突っ込みコメントをいただきました。

> テスト用データを作る際に、ランダムに分離していますが、
> 元のデータは時系列データにウィンドウをスライドしながら作ったものなので、
> テスト用データの中に、「学習データと1つずれただけ」のような、学習データと
> 部分的に被るデータが多数含まれる事になります。
> そのため、元のデータに*Momentum*が存在するとき、結果が不正確に高く評価されると思います。

なるほど涙

データをスライドしながら作成するまではともかく、学習と評価にそれを混ぜてしまっていました。

そうなると、部分的に全く一緒のデータを予測することになってしまい、学習で入力された結果と同じものをそのまま返すようになってしまっています。

これ、直に「値」を予測していれば、もっとすごく当たっているような、変な結果が導かれることになりそうですね。

となると予測問題とするには、完全に切り分けるか、長い間系列を予測させ続けるか、遠い先の値を予測するかといった形になるのだろうか。

時系列って難しいですね…。

タイトルも改め、「してみたかった」に変更しました。

コメントしていただいた方、ありがとうございました。

0

← Previous 統計検定準１級の試験対策について

次へ PyTorchで深層強化学習（DQN、DoubleDQN）を実装してみた →

“畳み込みニューラルネットワークで為替予測してみたかった” への 0 件のフィードバック

コメントを残す

メールアドレスが公開されることはありません。 * が付いている欄は必須項目です

コメント

名前 *

メールアドレス *

サイト

新しいコメントをメールで通知

新しい投稿をメールで受け取る

コメントを送信

Twitter

管理人執筆の本

たのしいベイズモデリング2

posted with ヨメレバ
豊田 秀樹/武藤 拓之 北大路書房 2019年11月18日

楽天ブックス

Amazon

アーカイブ

月を選択

カテゴリー

JavaScript (12)
Python (36)
R (1)
Ruby (2)
データ分析 (12)
データ可視化 (21)
強化学習 (4)
未分類 (10)
深層学習 (20)

まとめ投稿

Pythonによる仮説検定まとめ
Pythonによる区間推定まとめ
Rによる仮説検定まとめ
Rによる区間推定まとめ
強化学習の勉強でおすすめの書籍まとめ
時系列分析の勉強でおすすめの書籍まとめ
深層学習の勉強でおすすめの書籍まとめ
統計学や機械学習の勉強でおすすめの書籍まとめ
自然言語処理の勉強でおすすめの書籍まとめ

検索 …

スポンサーリンク

プライバシーポリシー

・掲載されている広告について

当サイトでは、第三者配信の広告サービス（Googleアドセンス）を利用しています。
このような広告配信事業者は、ユーザーの興味に応じた商品やサービスの広告を表示するため、
当サイトや他サイトへのアクセス情報 『Cookie』(氏名、住所、メール アドレス、電話番号は含まれません) を使用することがあります。
Googleアドセンスに関する詳細やこのような情報が広告配信事業者に使用されないようにする方法については、こちらをご参照ください。

・使用しているアクセス解析ツールについて

当サイトでは、Googleによるアクセス解析ツール「Googleアナリティクス」を利用しています。
このGoogleアナリティクスはトラフィックデータの収集のために『Cookie』を使用しています。
このトラフィックデータは匿名で収集されており、個人を特定するものではありません。
また『Cookie』を無効にすることで収集を拒否することが出来ますので、お使いのブラウザの設定をご確認ください。
この規約に関して、詳しくはこちらをご覧ください。