実装演習レポート
科目：深層学習

[Day 4]

【強化学習】
要点
・強化学習は長期的に報酬を最大化できるように環境の中で行動を選択できるエージェントを作ることを目標とする機械学習の一分野である。
・キャンペーンのコストという負の報酬と売上という正の報酬を最大化するために、行動を改善していく。
・過去のデータでベストとされる行動のみを常に取り続ければ他にもっとベストな行動を見つけることはできない（探索が足りない)。未知の行動のみを常に取り続ければ過去の経験が活かせない（利用が足りない)。探索と利用はトレードオフの関係性にある。
・教師あり教師なし学習ではデータに含まれるパターンを見つけ出して予測することが目標だが、強化学習では優れた方策を見つけることが目標である。
・Q 学習は行動価値関数を行動するごとに更新することにより学習を進める方法である。
・関数近似法は価値関数や方策関数を関数近似する手法のことである。
・価値関数には、ある状態の価値に注目する状態価値関数と状態と価値を組み合わせた価値に注目する行動価値関数の二種類がある。
・方策関数は方策ベースの強化学習手法においてある状態でどのような行動を取るのかという確率を与える関数のことである。
・方策反復法は方策をモデル化して最適化する手法である。

$$\theta^{(t+1)} = \theta^{(t)} + \epsilon \nabla J(\theta)$$

【AlphaGo】
要点
・AlphaGo Lee は ValueNet（19x19、49 チャンネル）という価値関数と PolicyNet（盤面特徴入力 19x19、48 チャンネル）という方策関数から成る。
・二次元のデータなので畳み込みして pooling する。ReLU 関数を用い、PolicyNet では活性化関数として SoftMax を通して出力する。
・ValueNet では Convolution の後に全結合をかけて TanH Layer（-∞～∞の値を-1~1 までに変換する layer）を通して出力する。
・線形の方策関数である RollOutPolicy や PolicyNet を教師あり学習で学習させる。
・モンテカルロ木探索で PlayOut と呼ばれるランダムシミュレーションを多数回行い、そ

の勝敗を集計して着手の優劣を決定する。
・AlphaGo Zero は教師あり学習を一切行わず、強化学習のみで作成する。Residual Net を導入している。ResidualNetwork はネットワークにショートカットを設ける。

【データ並列化】
要点
・分散深層学習とは、複数の計算資源（ワーカー）を使用し、並列的にニューラルネットを構成することでモデルを高速に学習させ、効率の良い学習を行うことである。
・深層学習は多くのデータを使用したり、パラメータ調整のために多くの時間を使用したりするため、データ並列化、モデル並列化、GPU による高速技術が不可欠である。
・データ並列化では親モデルを各ワーカーに子モデルとしてコピーし、データを分割しワーカーごとに計算させる。
・各モデルのパラメータの合わせ方で同期型か非同期型か決まる。同期型は各ワーカーの計算終了を待ち、全ワーカーの勾配が出たところで勾配の平均を計算して親モデルのパラメータを更新する。非同期型は各ワーカーの計算を待たず、子モデルごとに更新する。非同期型の方が処理スピードは早いが同期型の方が精度は良いことが多い。

【モデル並列化】
要点
・親モデルを各ワーカーに分割し、それぞれのモデルを学習させ、全てのデータで学習が終わった後で一つのモデルに復元化する。モデルを分割して並列に処理する。
・モデルが大きい時はモデル並列化を、データが大きい時はデータ並列化をすると良い。
・モデルのパラメータ数が多いほど、スピードアップの効率も向上する。
・Large Scale Distributed Deep Networks という Google 社が 2016 年に出した論文で紹介され、TensorFlow の前身といわれている。
・GPGPU (General-purpose of GPU)は元々の使用目的であるグラフィック以外の用途で使用される GPU の総称である。
・CPU は高性能なコアが少数あり、複雑で連続的な処理が得意である。
・GPU は比較的低性能なコアが多数あり、簡単な並列処理が得意である。ニューラルネットの学習は単純な行列演算が多いので高速化が可能である。
・CUDA は GPU 上で並列コンピューティングを行うためのプラットフォームであり、ＮＶＩＤＩＡ社が開発している GPU のみで使用可能である。Deep Learning 用に提供されている。
・OpenCL はオープンな並列コンピューティングのプラットフォームであり Intel, AMD,

ARM など NVIDIA 社以外の会社の GPU からでも使用可能である。

【量子化】

要点

・ネットワークが大きくなると大量のパラメータが必要となり学習や推論に多くのメモリと演算処理が必要となる。通常のパラメータの 64bit 浮動小数点を 32bit など下位の精度に落とすことでメモリと演算処理の削減を行う。少数の数字をコンピューターで表すことを量子化（Quantization）という。

・ニューロンの重みを浮動小数点の bit 数を少なくし有効桁数を下げることでニューロンのメモリサイズを小さくすることができ、多くのメモリを消費するモデルのメモリ使用量を抑え、省メモリ化することができる。

・32bit を用いて小数点を表すことを単精度、64bit を倍精度、16bit を半精度と呼ぶ。

・FLOPS は floating operations で、GPU の精度を表す単位である。

・モデルの精度が多少悪くなっても 16bit（半精度）の計算スピードで学習する方が現実的である。

【蒸留】

要点

・精度の高いモデルはニューロンの規模が大きなモデルになっているため、推論に多くのメモリと演算処理が必要である。規模の大きなモデルの知識を使い軽量なモデルの作成を行うことを蒸留という。

・学習済みの精度の高いモデルの知識を軽量なモデルへ継承させることをモデルの簡約化という。

・蒸留は教師モデル（予測精度の高い複雑なモデルやアンサンブルされたモデル）と生徒モデル（軽量化されたモデル）の 2 つで構成され、教師モデルの重みを固定し生徒モデルの重みを更新していく。誤差は教師モデルと生徒モデルのそれぞれの誤差を使い重みを更新していく。

・蒸留により少ない学習回数でより精度の良いモデルを作成することができる。

【プルーニング】

要点

・モデルの精度に寄与が少ないニューロンを削減することでモデルの軽量化、高速化を行うことをプルーニングという。重みが閾値以下のニューロンを削減する。

・重みの閾値を高くするとニューロンは大きく削減できるが精度も減少する。

【MobileNet】（画像認識ネットワーク）

要点

・MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications という論文でディープラーニングモデルの軽量化・高速化・高精度化を実現した。Depthwise Separable Convolution という手法を用いて計算量を軽減する。

・画像認識タスクに用いられるネットワークの軽量化を試みる。

・一般的な畳み込みレイヤーは入力特徴マップ（チャネル数）H x W x C、畳み込みカーエルサイズ K x K x C、出力チャネル数（フィルタ数）M、出力マップは H x W x M となる。ストライド 1 でパディングを適用した場合の畳み込み計算の計算量は H x W x K x K x C x M となる。MobileNets は Depthwise Convolution と Pointwise Convolution の組み合わせで軽量化を実現した。

・Depthwise Convolution は入力マップのチャネルごとに畳み込み（フィルタ数は 1 固定）を実施し、出力マップをそれらと結合する。出力マップは入力マップのチャネル数と同じになり、計算量は H x W x C x K x K となる。

・Pointwise Convolution は 1 x 1 x C のカーネルで畳み込みを入力マップのポイントごとに実施する。出力マップのチャネル数はフィルタ数分だけ作成可能で、計算量は H x W x C x M となる。

・Depthwise Convolution はチャネル毎に空間方向へ畳み込む。すなわち Dk x Dk x 1 のサイズのフィルターをそれぞれ用いて計算を行うため、その計算量は Dk x Dk x C x H x W

・Depthwise Convolution の出力を Pointwise Convolution によってチャネル方向に畳み込む。すなわち、出力チャネル毎に 1 x 1 x M サイズのフィルターをそれぞれ用いて計算を行うため、その計算量は H x W x C x M となる。

【DenseNet】（画像認識ネットワーク）

要点

・Densely Connected Convolutional Networks という論文で Dense Convolutional Network という畳み込みニューラルネットワークアーキテクチャの一種が紹介されている。

・Dense Block と呼ばれるモジュールを用いて前ブロックで計算した出力に入力特徴マップを足し合わせる。

・増加チャネル数 k を growth rate と呼び、k が大きくなるほどネットワークが大きくなるため、小さな整数に設定するのが良い。

・Dense block の間を convolution と pooling を行う transition layer でつなぎ、チャネル数

を減らす。
・DenseNet の DenseBlock では前方の各層からの出力全てが後方の層への入力として用いられる。ResNet の ResidualBlock では前 1 層の入力のみ後方の層へ入力される。
・DenseBlock では成長率（Growth Rate）と呼ばれるハイパーパラメータが存在する。ブロック毎に k 個ずつ特徴マップのチャネル数が増加していく時 k を成長率と呼ぶ。

【BatchNorm】（画像認識ネットワーク）
要点
・Bath Norm はミニバッチに含まれる sample の同一チャネルが同一分布に従うよう正規化し、Layer Norm はそれぞれの sample の全ての pixels が同一分布に従うよう正規化し、Instance Norm はさらに channel も同一分布に従うよう正規化する。
・Batch Size が小さい条件下では、学習が収束しないことがあり、代わりに Layer Normalization などの正規化手法が使われることが多い。
・H x W x C のサンプルが N 個あった場合に、N 個の同一チャネルが正規化の単位となる。ミニバッチのサイズを大きく取れない場合には効果が薄くなってします。

【LayerNorm】（画像認識ネットワーク）
要点
・Layer Norm は N 個のサンプルのうち一つに注目する。H x W x C の全ての pixel が正規化の単位となる。ミニバッチの数に依存しない。
・Layer Norm は入力データや重み行列に対して、以下の操作を施しても出力が変わらないことが知られている。
・Instance Norm は各サンプルのチャネル毎に正規化し、コントラストの正規化に寄与する。画像のスタイル転送やテクスチャ合成タスクなどで利用される。

【WaveNet】（音声生成モデル）
要点
・時系列データに対して畳み込み（Dilated convolution）を適用する。
・Dilated convolution は層が深くなるにつれて畳み込むリンクを離す。
・深層学習を用いて結合確率を学習する際に、効率的に学習できるアーキテクチャを提案したことが WaveNet の大きな貢献の一つである。提案された新しい Convolution 型アーキテクチャは Dilated causal convolution と呼ばれ、結合確率を効率的に学習できるようになっている。

・Dilated causal convolution を用いた際の大きな利点は、単純な Convolution layer と比べてパラメータ数に対する受容野が広い。

【Seq2seq】

要点

・Seq2seq は系列(Sequence)を入力として系列を出力するもので、Encoder-Decoder モデルとも呼ばれる。入力系列が Encode（内部状態に変換）され、内部状態から Decode（系列に変換）する。

・RNN（系列データを読み込むために再帰的に動作する NN）と言語モデル（単語の並びに確率を与えるモデル）の理解が重要である。

・RNN の目標は事後確率を求めることである。

【Transformer】

要点

・Attention は辞書オブジェクトであり、query に一致する key を索引し、対応する value を取り出す操作であると見なすことができる。

・Attention は文長が大きくなっても精度が落ちない。

・2017 年 6 月に Attention is all you need という論文で Transformer モデルが発表された。RNN を使わず、必要なのは Attention のみである。

・単語ベクトルに単語の位置を追加し、複数のヘッドで Dot Product Attention を行い、単語の位置ごとに独立処理する全結合をし、未来の単語を見ないようにマスクする。

・入力を全て同じにして学習的に注意箇所を決めていくのが Self-Attention の特徴である。

・Multi-Head attention は重みパラメタの異なる 8 個のヘッドを使用する。

・Decoder は Encoder と同じく 6 層だが、各層で二種類の注意機構がある。

・入出力の差分を学習させる Add（Residual Connection）（学習・テストエラーの低減）と各層においてバイアスを除く活性化関数への入力を平均 0、分散 1 に正則化する Norm（Layer Normalization）（学習の高速化）を実装してある。

【物体検知】

要点

・代表的なデータセットとして VOC12、ILSVRC17、MS COCO18、OICOD18 などがある。BOX/画像が大きい方がより実生活に則する。

・分類問題における評価指標は Confusion Matrix（混同行列）から Precision　TP/(TP+FP)

やRecall TP/(TP+FN)として読み取る。Confidenceの閾値を変化させても confusion matrixに入るサンプル数の合計数は変化しないが、物体検出の場合はサンプル数自体が異なってくる。

・物体検出においてはクラスラベルだけでなく物体位置の予測精度の指標として IoU (Intersection over Union)が導入された。別名 Jaccard 係数とも呼ばれ、IoU = TP / (TP + FP + FN)で表される。

・AP (Average Precision)は confidence の閾値をβとするとき、Recall = R (β), Precision = P(β)となるので P = f(R) [Precision-Recall curve]であることから、$AP = \int_0^1 P(R)dR$（PR曲線の下側面積）となる。クラス数が C の時、$mAP = \frac{1}{C}\sum_{i=1}^{C} AP_i$となる。

・MS COCO で導入された指標は 0.5 で固定していた IoU を 0.5 から 0.95 まで 0.05 刻みで AP&mAP を計算し算術平均を計算する。

$$mAP_{COCO} = \frac{mAP_{0.5} + mAP_{0.55} + \cdots + mAP_{0.95}}{10}$$

・最近は FPS (Flames per Second)という 1 秒間あたりの処理フレーム数（検出速度）も指標として利用される。

・inference time (ms)は 1 推論にかかる時間を示す指標である。

・2012 年の AlexNet の登場により SIFT から DCNN に移行した。

・2 段階検出器(Two-stage detector)は候補領域の検出とクラス推定を別々に行うので相対的に精度が高いが計算量が大きく推論も遅い傾向がある。

・ 1 段階検出器(One-stage detector)は候補領域の検出とクラス推定を同時に行うので相対的に精度が低いが計算量が小さく推論も早い傾向にある。

・SSD (Single Shot Detector)は VGG16 を改良したネットワークアーキテクチャーである。

【セグメンテーション】

<u>要点</u>

・Semantic Segmentation では Convolution や Pooling により解像度が落ちていく。どのようにして元の解像度に戻すのかが課題であり、元の解像度に戻す手法を up-sampling という。

・Deconvolution/Transposed convolution では特徴マップの pixel 間隔を stride だけ空け、特徴マップの周りに(kernel size -1) -padding だけ余白を作り畳み込み演算を行う。

・pooling によりローカルな情報 (輪郭等) が失われていくので、底レイヤーpooling 層の出力を element-wise addition することでローカルな情報を補完してから up-sampling する。

・pooling 時の位置情報を保持しておき(switch variables)、unpooling を行い、解像度を上げることができる。

・pooling は受容野を広げるために必要だが、Convolution の段階で受容野を広げる工夫として dilated convolution という手法が用いられる。

【為替データの扱い】

要点

・為替の始値、高値、安値、終値のデータを読み込ませ、ミニバッチ処理（ミニバッチ数 32）をして終値を予測するモデルを作成する。

・モデルは LSTM を利用し、optimizer として Adam を使用し、エポック数 30 の学習をさせている。

演習問題レポート

USD/JPY LSTM(TensorFlow)

```
[1] import pandas as pd
    import numpy as np
    import tensorflow as tf
    import matplotlib.pyplot as plt
    % matplotlib inline
    import seaborn as sns
    sns.set()
```

データ読み込み

```
!mkdir data
!wget https://dl.dropbox.com/s/rzzcc89lokvzu3r/usd_jpy.csv -O data/usd_jpy.csv
```

```
--2021-01-02 12:40:27-- https://dl.dropbox.com/s/rzzcc89lokvzu3r/usd_jpy.csv
Resolving dl.dropbox.com (dl.dropbox.com)... 162.125.1.15, 2620:100:6016:15::a27d:10f
Connecting to dl.dropbox.com (dl.dropbox.com)|162.125.1.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://dl.dropboxusercontent.com/s/rzzcc89lokvzu3r/usd_jpy.csv [following]
--2021-01-02 12:40:27-- https://dl.dropboxusercontent.com/s/rzzcc89lokvzu3r/usd_jpy.csv
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 162.125.1.15, 2620:100:6022:15::a27d:420f
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)|162.125.1.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 107599 (105K) [text/csv]
Saving to: 'data/usd_jpy.csv'

data/usd_jpy.csv    100%[===================>] 105.08K  --.-KB/s    in 0.02s

2021-01-02 12:40:28 (5.87 MB/s) - 'data/usd_jpy.csv' saved [107599/107599]
```

```
[3] df = pd.read_csv('data/usd_jpy.csv')
    df.head()
```

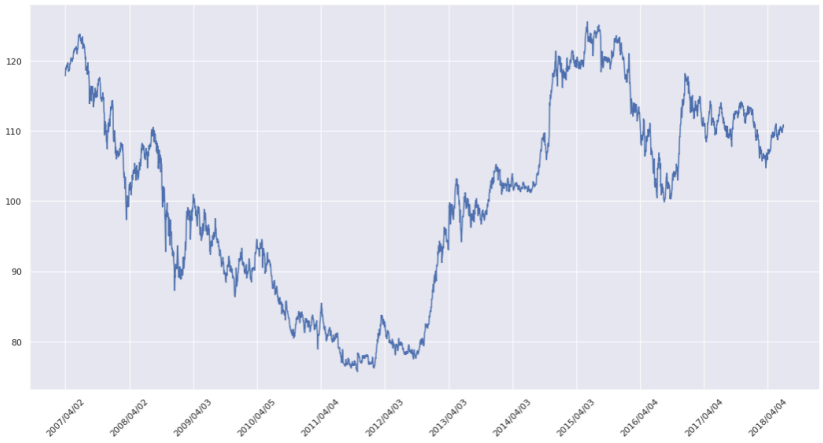
	日付	始値	高値	安値	終値
0	2007/04/02	117.84	118.08	117.46	117.84
1	2007/04/03	117.84	118.98	117.72	118.96
2	2007/04/04	118.92	119.08	118.56	118.72
3	2007/04/05	118.72	118.99	118.44	118.72
4	2007/04/06	118.72	119.39	118.67	119.27

```
[4] df.describe()
```

	始値	高値	安値	終値
count	2925.000000	2925.000000	2925.000000	2925.000000
mean	100.618236	101.057046	100.119538	100.615289
std	13.849288	13.881241	13.800449	13.851496
min	75.760000	75.980000	75.570000	75.680000
25%	89.690000	90.210000	89.120000	89.740000
50%	102.080000	102.440000	101.720000	102.090000
75%	111.700000	112.140000	111.170000	111.700000
max	125.660000	125.860000	124.540000	125.550000

[5]
```
plt.figure(figsize = (18,9))
plt.plot(np.arange(df.shape[0]), df['終値'].values)
plt.xticks(np.arange(0, df.shape[0], 260), df['日付'].loc[::260], rotation=45)
```

```
([<matplotlib.axis.XTick at 0x7f6746f110f0>,
  <matplotlib.axis.XTick at 0x7f6746f110b8>,
  <matplotlib.axis.XTick at 0x7f6746f05cc0>,
  <matplotlib.axis.XTick at 0x7f6746ebaeb8>,
  <matplotlib.axis.XTick at 0x7f6746ed3390>,
  <matplotlib.axis.XTick at 0x7f6746ed3828>,
  <matplotlib.axis.XTick at 0x7f6746ed3cc0>,
  <matplotlib.axis.XTick at 0x7f6746edc208>,
  <matplotlib.axis.XTick at 0x7f6746edc630>,
  <matplotlib.axis.XTick at 0x7f6746edcac8>,
  <matplotlib.axis.XTick at 0x7f6746edcf60>,
  <matplotlib.axis.XTick at 0x7f6746ee4438>],
 [Text(0, 0, '2007/04/02'),
  Text(0, 0, '2008/04/02'),
  Text(0, 0, '2009/04/03'),
  Text(0, 0, '2010/04/05'),
  Text(0, 0, '2011/04/04'),
  Text(0, 0, '2012/04/03'),
  Text(0, 0, '2013/04/03'),
  Text(0, 0, '2014/04/03'),
  Text(0, 0, '2015/04/03'),
  Text(0, 0, '2016/04/04'),
  Text(0, 0, '2017/04/04'),
  Text(0, 0, '2018/04/04')])
```



[6]
```
close_prices = df['終値'].values

train_ratio = 0.8
n_train = int(len(close_prices) * train_ratio)
train, test = close_prices[:n_train], close_prices[n_train:]
train = train.reshape(-1, 1)
test = test.reshape(-1, 1)
```

[7]
```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
train = scaler.fit_transform(train)
test = scaler.transform(test)
```

[8]
```
print(np.min(train))
print(train[32])
print(np.max(train))
```

```
0.0
[0.9051534]
1.0000000000000002
```

[9]
```
print(len(train))
```

```
2340
```

```
import math

class DataGenerator(tf.keras.utils.Sequence):
    def __init__(self, data, batch_size, n_steps, input_size, output_size):
        self.data = data
        self.batch_size = batch_size
        self.n_steps = n_steps
        self.input_size = input_size
        self.output_size = output_size

    def __len__(self):
        return math.floor((len(self.data) - self.n_steps) / self.batch_size)

    def __getitem__(self, idx):
        xs = np.zeros((self.batch_size, self.n_steps, self.input_size))
        ys = np.zeros((self.batch_size, self.output_size))

        for i in range(self.batch_size):
            step_begin = (idx * self.batch_size) + i
            step_end = (idx * self.batch_size) + i + self.n_steps
            x = np.zeros((self.n_steps, self.input_size))
            y = np.zeros((self.output_size))

            if step_end >= len(self.data):
                break
            else:
                x = self.data[step_begin: step_end]
                y = self.data[step_end]

            xs[i] = x
            ys[i] = y

        return xs, ys
```

```
[11] # 学習設定
     batch_size = 32     # ミニバッチサイズ
     n_steps = 50        # 入力系列の長さ
     input_size = 1      # 入力の次元
     hidden_size = 50    #  中間層の次元
     output_size = 1     #  出力層の次元

     n_epochs = 30       # エポック数
```

```
[12] train_gen = DataGenerator(train, batch_size, n_steps, input_size, output_size)
```

モデル構築

```
def create_model():
    inputs = tf.keras.layers.Input(shape=(n_steps, input_size))
    lstm = tf.keras.layers.LSTM(hidden_size)(inputs)
    output = tf.keras.layers.Dense(output_size)(lstm)

    model = tf.keras.Model(inputs, output)
    model.compile(optimizer='adam', loss='mse', metrics='accuracy')

    return model

model = create_model()

model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 50, 1)]           0
_________________________________________________________________
lstm (LSTM)                  (None, 50)                10400
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
```

学習

```
[14] model.fit_generator(train_gen, epochs=30, shuffle=True)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/30
71/71 [==============================] - 8s 5ms/step - loss: 0.1335 - accuracy: 7.7398e-04
Epoch 2/30
71/71 [==============================] - 0s 5ms/step - loss: 0.0017 - accuracy: 2.5937e-04
Epoch 3/30
71/71 [==============================] - 0s 5ms/step - loss: 0.0010 - accuracy: 2.4651e-04
Epoch 4/30
71/71 [==============================] - 0s 5ms/step - loss: 8.0466e-04 - accuracy: 2.4848e-04
Epoch 5/30
71/71 [==============================] - 0s 5ms/step - loss: 8.5875e-04 - accuracy: 2.3021e-04
Epoch 6/30
71/71 [==============================] - 0s 5ms/step - loss: 7.1670e-04 - accuracy: 0.0011
Epoch 7/30
71/71 [==============================] - 0s 5ms/step - loss: 7.6029e-04 - accuracy: 5.0284e-04
Epoch 8/30
71/71 [==============================] - 0s 5ms/step - loss: 8.3795e-04 - accuracy: 8.8271e-04
Epoch 9/30
71/71 [==============================] - 0s 5ms/step - loss: 8.6096e-04 - accuracy: 0.0017
Epoch 10/30
71/71 [==============================] - 0s 5ms/step - loss: 6.1518e-04 - accuracy: 3.8650e-04
Epoch 11/30
71/71 [==============================] - 0s 5ms/step - loss: 6.7720e-04 - accuracy: 0.0010
Epoch 12/30
71/71 [==============================] - 0s 5ms/step - loss: 7.6738e-04 - accuracy: 0.0013
Epoch 13/30
71/71 [==============================] - 0s 5ms/step - loss: 8.1626e-04 - accuracy: 2.9038e-04
Epoch 14/30
71/71 [==============================] - 0s 5ms/step - loss: 5.3409e-04 - accuracy: 7.5928e-04
Epoch 15/30
71/71 [==============================] - 0s 5ms/step - loss: 5.1653e-04 - accuracy: 0.0011
Epoch 16/30
71/71 [==============================] - 0s 4ms/step - loss: 5.2009e-04 - accuracy: 1.9584e-04
Epoch 17/30
71/71 [==============================] - 0s 5ms/step - loss: 7.1985e-04 - accuracy: 0.0013
Epoch 18/30
71/71 [==============================] - 0s 5ms/step - loss: 7.3180e-04 - accuracy: 0.0014
Epoch 19/30
71/71 [==============================] - 0s 4ms/step - loss: 6.7007e-04 - accuracy: 8.4392e-04
Epoch 20/30
71/71 [==============================] - 0s 4ms/step - loss: 5.0046e-04 - accuracy: 0.0013
Epoch 21/30
71/71 [==============================] - 0s 5ms/step - loss: 5.6088e-04 - accuracy: 0.0015
Epoch 22/30
71/71 [==============================] - 0s 4ms/step - loss: 5.5656e-04 - accuracy: 7.1724e-04
Epoch 23/30
71/71 [==============================] - 0s 5ms/step - loss: 4.8900e-04 - accuracy: 4.3450e-04
Epoch 24/30
71/71 [==============================] - 0s 5ms/step - loss: 5.6935e-04 - accuracy: 9.8271e-04
Epoch 25/30
71/71 [==============================] - 0s 5ms/step - loss: 5.4820e-04 - accuracy: 0.0015
Epoch 26/30
71/71 [==============================] - 0s 4ms/step - loss: 4.5423e-04 - accuracy: 0.0024
Epoch 27/30
71/71 [==============================] - 0s 5ms/step - loss: 4.4101e-04 - accuracy: 1.1695e-04
Epoch 28/30
71/71 [==============================] - 0s 5ms/step - loss: 4.1419e-04 - accuracy: 4.9865e-04
Epoch 29/30
71/71 [==============================] - 0s 5ms/step - loss: 4.9706e-04 - accuracy: 9.3109e-04
Epoch 30/30
71/71 [==============================] - 0s 5ms/step - loss: 5.6106e-04 - accuracy: 0.0013
<tensorflow.python.keras.callbacks.History at 0x7f673016cc18>
```

テスト

```
[15] test_gen = DataGenerator(test, batch_size, n_steps, input_size, output_size)
     predicts = model.predict_generator(test_gen)
```

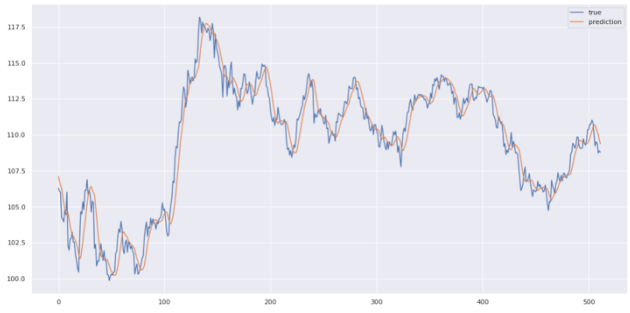
```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/training.py:1905: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  warnings.warn('`Model.predict_generator` is deprecated and '
```

編集するにはダブルクリックするか Enter キーを押してください

```
[16] fixed_predicts = scaler.inverse_transform(predicts)
     fixed_test = scaler.inverse_transform(test)
```

```
# テストデータに対する予測を可視化
mod = (len(fixed_test) - n_steps) % batch_size
xx = np.arange(len(fixed_predicts))
plt.figure(figsize=(18, 9))
plt.plot(xx, fixed_test[n_steps:-mod], label='true')
plt.plot(xx, fixed_predicts, label='prediction')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f66db448908>
```



為替データを読み込んで予測できるようになった。

https://study-ai.com/jdla/