but humans create 👒, discover 🚀 and love ❤️

Jun 6, 2017 · 6 min read

Neural networks for algorithmic trading. Multivariate time series

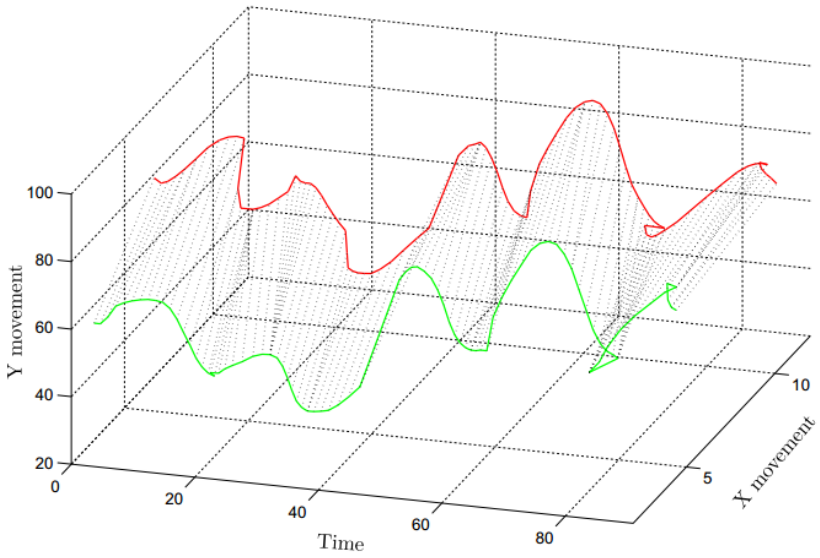


Illustration of 2-variate time series

In previous post we discussed several ways to forecast financial time series: how to normalize data, make prediction in the form of real value or binary variable and how to deal with overfitting on highly noisy data. But what we skipped (on purpose)—is that our .csv file with prices basically has much more data that we may use. In last post only close prices with some transformation were used, but what can happen if we will consider also high, low, open prices and volume of every historical day? This leads us to working with multidimensional, e.g. **multivariate time series**, where on every time stamp we have more than just one variable—in our case we will work with whole OHLCV tuple.

Other posts:

1. Simple time series forecasting (and mistakes done)

2. Correct 1D time series forecasting + backtesting
3. Multivariate time series forecasting
4. Volatility forecasting and custom losses
5. Multitask and multimodal learning
6. Hyperparameters optimization
7. Enhancing classical strategies with neural nets
8. Probabilistic programming and Pyro forecasts

In this article we will see how to preprocess multivariate time series, in particular, what to do with every dimension, how to define and train a neural network on this kind of data and will compare results with what we had in last post.

As always, you can jump directly to the code.

Data preparation

To understand better what multidimensional time series is, let's remember how look images, that in fact also have not just two dimensions (height and width), but also "depth" that represents color channels:

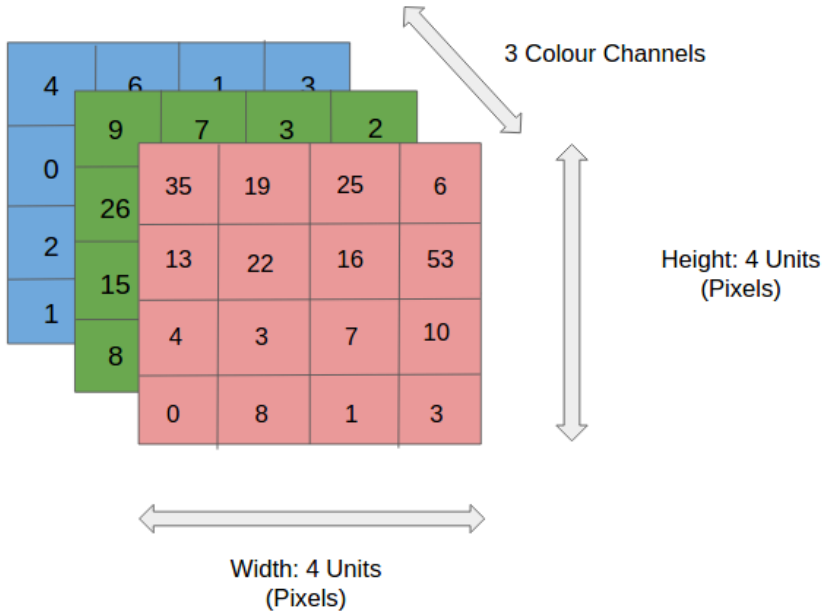


Image channels and dimensions

In case of time series, our image is just 1D (the plot we usually see on the graph) and the role of channels play different values—open, high, low, close prices and volume of operations. You can also think about it from other point of view—on any time stamp our time series is represented not with a single value, but with a vector (open, high, low, close prices and volume of every day), but metaphor with images is more useful to understand why we will apply convolutional neural networks to this problem today.

One of the most important moment about multivariate time series—the dimensions can come from different sources, can have different nature and can be totally uncorrelated and have different distribution, so we have to normalize them independently! We will use an ugly, but more or less adequate trick from last post:

> *We don't need to predict some exact value, so expected value and variance of the future isn't very interesting for us—we just need to predict the movement up or down. That's why we will risk and normalize our 30-days windows only by their mean and variance (z-score normalization), supposing that just during single time window they don't change much and not touching information from the future.*

But we are going to normalize every dimension of time window independently:

```
for i in range(0, len(data_original), STEP):
    try:
        o = openp[i:i+WINDOW]
        h = highp[i:i+WINDOW]
        l = lowp[i:i+WINDOW]
        c = closep[i:i+WINDOW]
        v = volumep[i:i+WINDOW]

        o = (np.array(o) - np.mean(o)) / np.std(o)
        h = (np.array(h) - np.mean(h)) / np.std(h)
        l = (np.array(l) - np.mean(l)) / np.std(l)
        c = (np.array(c) - np.mean(c)) / np.std(c)
        v = (np.array(v) - np.mean(v)) / np.std(v)
```

But as we want to forecast movement of a price up or down next day, we need to consider the change of a single dimension:

```
x_i = closep[i:i+WINDOW]
y_i = closep[i+WINDOW+FORECAST]
```

```
last_close = x_i[-1]
    next_close = y_i
```

```
if last_close < next_close:
    y_i = [1, 0]
else:
    y_i = [0, 1]
```

So, the data we will train on—are time windows of, like before, 30 days, but now on every day we will consider **whole OHLCV data correctly normalized** to predict the direction of close price movement. Full code for data preparation and neural network training you can find here.

Neural network architecture

As I mentioned before, I would like to use CNN as a classifier. Mainly I choose it because of flexibility and interpretability of hyperparameters (convolutional kernel, downsampling size etc) and performance similar to RNNs, better than MLP with much faster training.

The code for our network for today looks like:

```
model = Sequential()
model.add(Convolution1D(input_shape = (WINDOW, EMB_SIZE),
                        nb_filter=16,
                        filter_length=4,
                        border_mode='same'))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dropout(0.5))
```

```
model.add(Convolution1D(nb_filter=8,
                        filter_length=4,
                        border_mode='same'))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dropout(0.5))
```

```
model.add(Flatten())
```

```
model.add(Dense(64))
model.add(BatchNormalization())
model.add(LeakyReLU())
```

```
model.add(Dense(2))
model.add(Activation('softmax'))
```

The only difference from an architecture from a very first post is changing the *EMB_SIZE* variable to 5 in our case.

Training process

Let's compile the model:

```
opt = Nadam(lr=0.002)
```

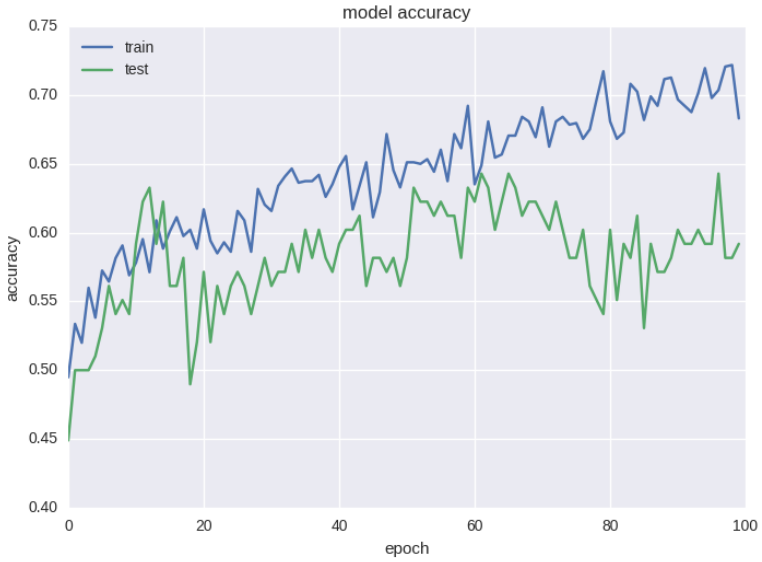
```
reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.9,
patience=30, min_lr=0.000001, verbose=1)
checkpointer = ModelCheckpoint(filepath="model.hdf5",
verbose=1, save_best_only=True)
```

```
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history = model.fit(X_train, Y_train,
          nb_epoch = 100,
          batch_size = 128,
          verbose=1,
          validation_data=(X_test, Y_test),
          callbacks=[reduce_lr, checkpointer],
          shuffle=True)
```

And check performance:

Loss after 100 epochs



Accuracy of binary classification after 100 epochs

From the plots we can clearly see that network trained adequately (for very noisy data), the loss of training set was decreasing with time while accuracy—increasing. And, what's the most important, comparing to univariate time series from previous post **we improved the performance from 58% to almost 65% of accuracy!**

To check overfitting we can also plot confusion matrix:

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

model.load_weights("model.hdf5")
pred = model.predict(np.array(X_test))

C = confusion_matrix([np.argmax(y) for y in Y_test],
[np.argmax(y) for y in pred])

print C / C.astype(np.float).sum(axis=1)
```

and we will get:

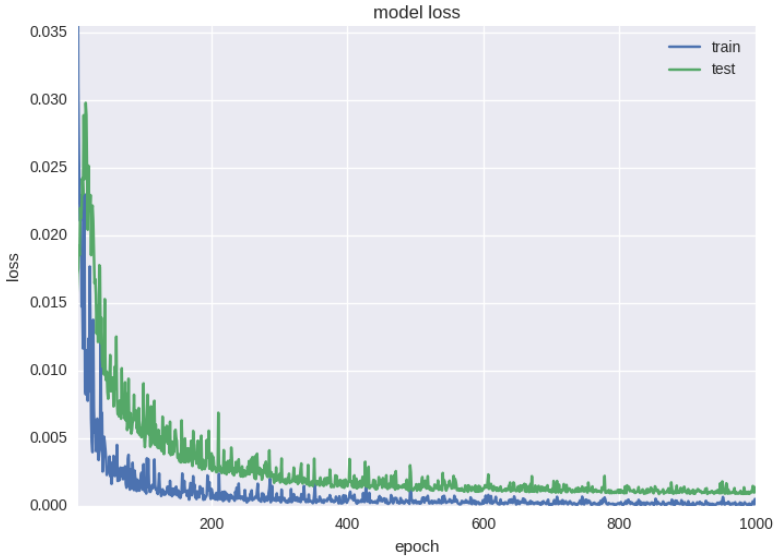
```
[[ 0.75510204  0.24489796]
[ 0.46938776  0.53061224]]
```

which shows that we predict “UP” movement with 75% of accuracy and “DOWN” with 53% of accuracy and this results of course can be balanced for the test dataset.

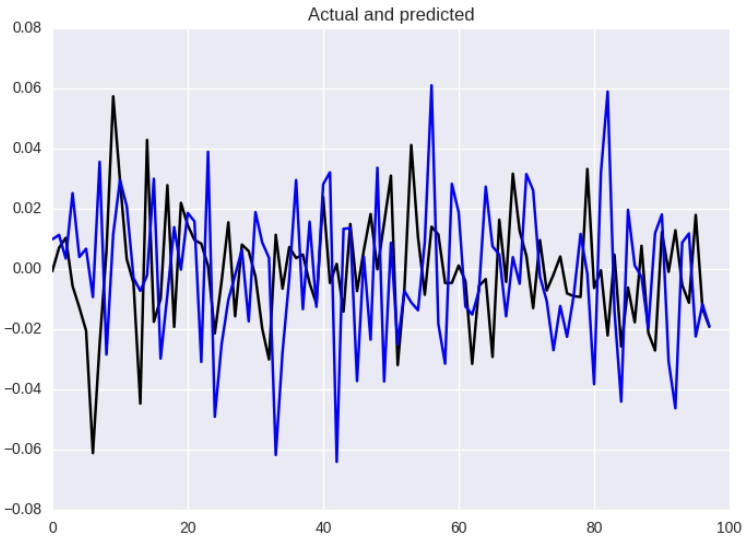
What about regression?

Instead of predicting the binary variable, we can predict the real value —next day return or close price. In our previous experiments we didn’t succeed to produce good results.

Unfortunately, for returns it still works bad:

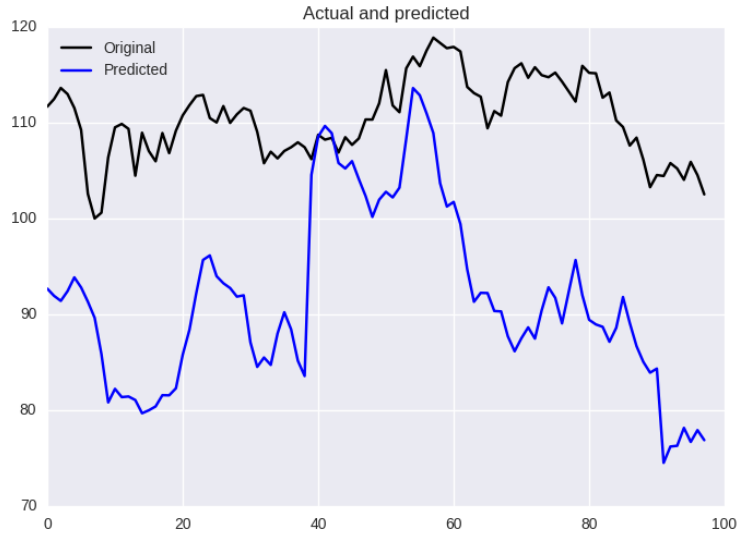


Loss decreasing for regression problem



Prediction of the price change

For prediction of value of close price the situation isn't better:



Prediction of close price

I am still trying different things for regression problem in financial data (like custom loss functions), if you have some suggestions, I'd like to discuss them in comments or PM.

Conclusions

We discussed the general pipeline of data preparation and normalization in case of multivariate time series, trained a CNN on them and we can report **significant (+7%) improvement** of classification problem—predicting if stock price will go up or down next day. Don't forget to check the full code and run it on your machine!

Meanwhile we still can state that regression problem is still too complicated for us and we will work on it later, choosing correct loss metrics and activation functions.

In next post I would like to introduce the concept of multimodal learning and we will use parameters not just from our .csv file with OHLCV tuples, but much more interesting things.

Stay tuned!