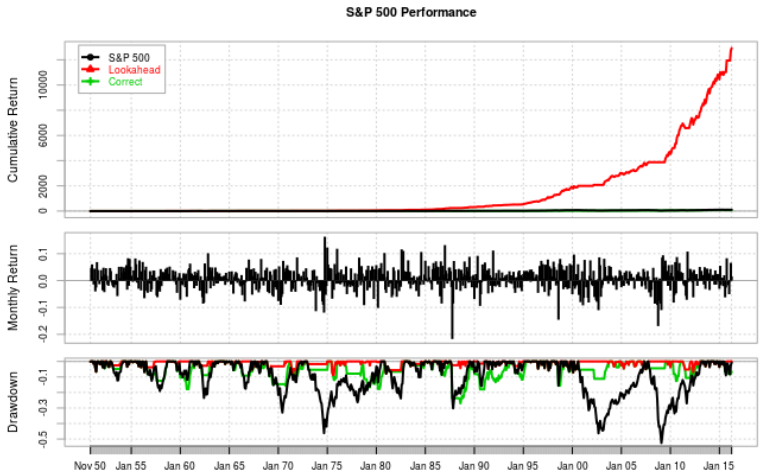
but humans create 👒, discover 🚀 and love ❤️ ike 👍,

Oct 19, 2017 · 6 min read

Neural networks for algorithmic trading: enhancing classic strategies



Hello everyone! In five last tutorials we were discussing financial forecasting with artificial neural networks where we compared different architectures for financial time series forecasting, realized how to do this forecasting adequately with correct data preprocessing and regularization, performed our forecasts based on multivariate time series and could produce really nice results for volatility forecasting and implemented custom loss functions. In the last one we have set and experiment with using data from different sources and solving two tasks with single neural network and optimized hyperparameters for better forecasts.

Today I want to make a sort of conclusion of financial time series with a practical forecasting use case: **we will enhance a classic moving average strategy with neural network and show that it really**

improves the final outcome and review new forecasting objectives you most probably would like to play with.

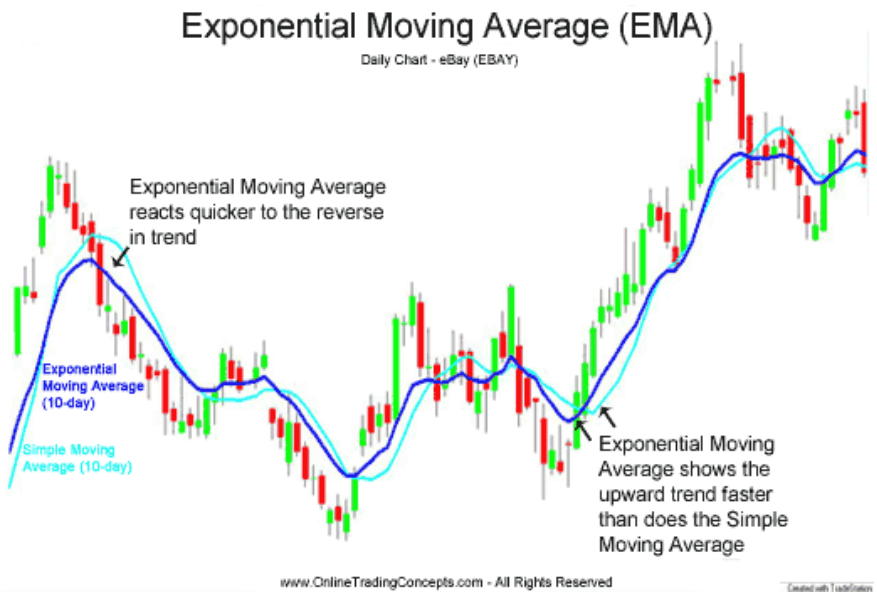
Previous posts:

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

You can check the code for training the neural network on my Github.

Main idea

We already have seen before, that we can forecast very different values —from price changes to volatility. Before we were considering these forecasts as something kind of abstract and even tried to trade just looking on these “up-down” predictions, which wasn’t that good. But we also know, that there are a lot of other trading strategies, that are based on technical analysis and financial indicators. For example, we can build moving averages of different window (one “long”, let’s say 30 days, and one more “short”, probably, 14 days) and we believe that crossing points are the moments where the trend changes:



Example of two moving averages crossing

But this trading strategy has one main pitfall: on the flat regions we will still do the trades on the points where nothing actually changes, so we will lose money:

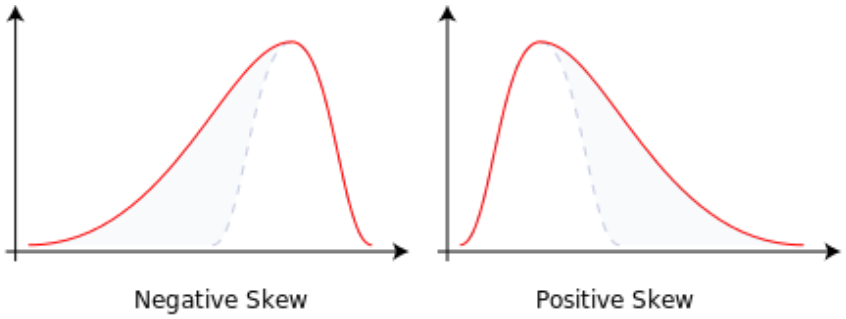
Example of flat region where moving averages are crossing

How we can overcome this with use of machine learning?

> *Let's check the following strategy hypothesis: on the moments where moving averages are crossing we will make the forecast of change of some characteristic, and if we really expect a jump, we will*

believe this trading signal. Otherwise, we will skip it, because we don't want to lose money on flat regions.

As a forecast objective I want to try skewness—a measure of asymmetry of a distribution. Let us assume, that if we forecast a change in a distribution it will mean that our current trend (not only flat region) will change in the future.



Skewness of distribution

Input data

Here we will use pandas and PyTi to generate more indicators to use them as input as well. We will use MACD, Ichimocku cloud, RSI, volatility and others. All these values will form multivariate time series which will be flatten for later use in MLP or will stay for CNN or RNN.

```
nine_period_high = pd.rolling_max(pd.DataFrame(highp),
window= ROLLING / 2)
nine_period_low = pd.rolling_min(pd.DataFrame(lowp), window=
ROLLING / 2)
ichimoku = (nine_period_high + nine_period_low) /2
ichimoku = ichimoku.replace([np.inf, -np.inf], np.nan)
ichimoku = ichimoku.fillna(0.).values.tolist()
```

```
macd_indie =
moving_average_convergence(pd.DataFrame(closep))
```

```
wpr = williams_percent_r(closep)
rsi = relative_strength_index(closep,  ROLLING / 2)
```

```
volatility1 =
pd.DataFrame(closep).rolling(ROLLING).std().values#.tolist()
volatility2 =
pd.DataFrame(closep).rolling(ROLLING).var().values#.tolist()
```

```
volatility = volatility1 / volatility2
volatility = [v[0] for v in volatility]
```

```
rolling_skewness =
pd.DataFrame(closep).rolling(ROLLING).skew().values
rolling_kurtosis =
pd.DataFrame(closep).rolling(ROLLING).kurt().values
```

Obtained indicator features I concatenate with OHLCV tuples to generate final vector.

Network architecture

Here I want to show one of the option how to train regularized MLP for time series forecasting:

```
main_input = Input(shape=(len(X[0]), ), name='main_input')
x = GaussianNoise(0.05)(main_input)
x = Dense(64, activation='relu')(x)
x = GaussianNoise(0.05)(x)
output = Dense(1, activation = "linear", name = "out")(x)
```

```
final_model = Model(inputs=[main_input], outputs=[output])
opt = Adam(lr=0.002)
final_model.compile(optimizer=opt, loss='mse')
```

> “Novel” point here is adding small noise to the input and to the output of the single layer of our neural network. It can work very similar to L2 regularization, mathematical explanation you can check in this amazing book.

We now assume that with each input presentation we also include a random perturbation $\epsilon_{\boldsymbol{W}} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \eta \boldsymbol{I})$ of the network weights. Let us imagine that we have a standard l-layer MLP. We denote the perturbed model as $\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x})$. Despite the injection of noise, we are still interested in minimizing the squared error of the output of the network. The objective function thus becomes

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x}, y, \boldsymbol{\epsilon W})} \left[(\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) - y)^2 \right] \tag{7.31}$$

$$= \mathbb{E}_{p(\boldsymbol{x}, y, \boldsymbol{\epsilon W})} \left[\hat{y}^2_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) - 2y \hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) + y^2 \right]. \tag{7.32}$$

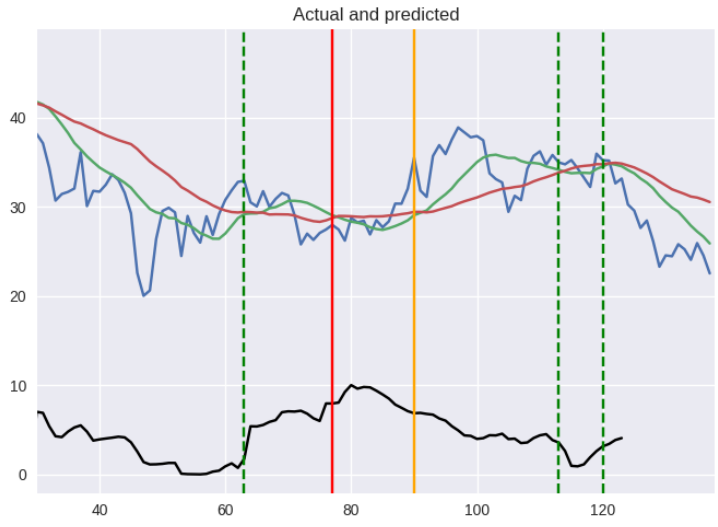
For small η, the minimization of J with added weight noise (with covariance $\eta \boldsymbol{I}$) is equivalent to minimization of J with an additional regularization term: $\eta \mathbb{E}_{p(\boldsymbol{x}, y)} \left[\|\nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\|^2 \right]$. This form of regularization encourages the parameters

Sample from http://www.deeplearningbook.org/contents/regularization.html

Neural network is trained in a usual way, let's check how our forecasts of skewness can improve (or no) the moving averages strategy.

We train our network on AAPL prices from 2012 to 2016 and as test on 2016–2017 as we did in one of previous tutorials.

After training of a network I have plotted our close prices, moving averages and vertical lines on crossing points: red and orange lines represent points where we would like to trade and green ones—where we better don't. It doesn't look perfect, but let's do backtesting to judge it.

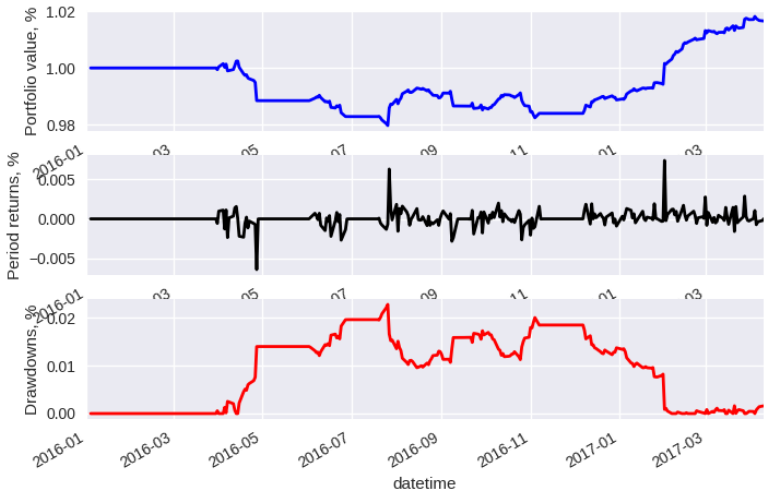


What moving average intersection is useful?

Results without neural network

I used backtesting described in this post, so I will provide just key metrics and plots:

[('Total Return', '1.66%'),
 ('Sharpe Ratio', '16.27'),
 ('Max Drawdown', '2.28%'),
 ('Drawdown Duration', '204')]
Signals: 9
Orders: 9
Fills: 9



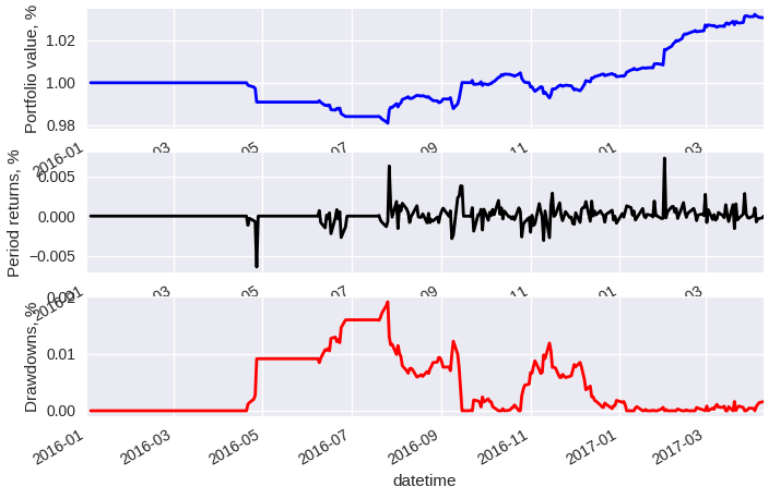
Results of backtesting of a rolling mean strategy

Results with neural network

How we will use only "red" and "orange" trading signals and skip the green ones. As we can see, this such a strategy did 2 trades less and it helped us to reduce the first drawdown a bit and increase final return almost twice!

[('Total Return', '3.07%'),
 ('Sharpe Ratio', '27.99'),
 ('Max Drawdown', '1.91%'),
 ('Drawdown Duration', '102')]

Signals: 7
Orders: 7
Fills: 7



Results of backtesting of a strategy with use of NN

Possible improvements

Seems like this idea at least has some sense! I would like to introduce you some possible improvements I highly recommend you to try by your own:

- Different indicator strategies: MACD, RSI
- Pairs trading strategies can be optimized extremely well with approach proposed
- Try to forecast different time series characteristics: Hurst exponent, autocorrelation coefficient, maybe other statistical moments

With this post I would like to finish (at least for a while) financial time series forecasting topic using neural networks. Let's be honest, it's definitely not a Holy Graal and we can't use them directly to predict if price will go up or down to make a lot of money. We considered different data sources and objectives, dealt carefully with overfitting and optimized hyperparameters. What conclusions we can do?

- Be careful about overfitting! You will do it in 99% of cases, don't trust values as 80% of accuracy of very nice looking plots—it must be a mistake
- Try to forecast something different but close prices or returns—volatility, skewness, maybe other characteristics
- Use multimodal learning if you have different data sources
- Don't forget to find the right hyperparameters!
- Create a strategy that can be a mix of some classical and based on machine learning and backtest it!

I hope that this series of posts was useful to someone, I will come back rather soon with news topics… Stay tuned! :)