Stock Market Short-Term Prediction

BY TOMAS UKKONEN
Novel Insight Research
tomas.ukkonen@novelinsight.fi

Abstract

This is second attemp to do stock market prediction. The first attempt by trying to use historical price data and deep 20-layer residual neural net failed because minimal neural network (1-layer) gives similar thought a bit less reliable results. Bayesian Neural Networks used together with Hamiltonian Monte Carlo sampling does not give very accurate results when predicting uncertainty in the next day stock market data. Further attempt to do 10 or more probabilistic computations of neural net's weights and using Markov consensus vote counting to combine results seem to be a bit useful. To get better results, more complex method will be attempted.

1 Bayesian Neural Network Differential Equation Model

Bayesian methods are used to learn differential equation model from stock market data. Probabilistic methods to handle uncertainty is used because differential equations are sensitive to noise in signals. We attempt to learn non-linear neural net based differential equation: $\frac{\partial \boldsymbol{x}(t)}{\partial t} = f(\boldsymbol{x}(t), \boldsymbol{w})$. The future values of $\boldsymbol{x}(t)$ are simulated using initial conditions $\boldsymbol{x}(0)$ and Runge-Kutta simulation method. These values are compared to measurements $\boldsymbol{x}_h(t)$ (historical data from the stock market). It is difficult or impossible to calculate gradient of the neural net, therefore sampling methods are attemped. We assume normally distributed error and minimize squared error. For efficient Hamiltonian Monte Carlo (HMC) sampling, we need also some kind of gradient of the squared error term.

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_i \|\boldsymbol{x}(t_i) - \boldsymbol{x}_h(t_i)\|^2 + \varepsilon,$$

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \sum_i \left(\int_0^{t_i} (\boldsymbol{x}(t_i) - \boldsymbol{x}_h(t_i))^T \nabla_{\boldsymbol{w}} f(\boldsymbol{x}(t), \boldsymbol{w}) d\, t\right)$$

Here we can use Runge-Kutta again to estimate gradient term: $\frac{\partial(\nabla_{\boldsymbol{w}} \boldsymbol{x}(t))}{\partial t}$. Also the weight vector is assumed to be normally distributed so we get a posterior $p(\boldsymbol{w}||\mathbf{data})$ from which we can sample.

1.1 Runge-Kutta for Jacobian/Gradient

In practice, we need to implement Runge-Kutta to estimate integrated gradient terms $\frac{\partial((\boldsymbol{x}(t_i) - \boldsymbol{x}_h(t_i))^T \nabla_{\boldsymbol{w}} \boldsymbol{x}(t_i))}{\partial t} = \frac{\partial \boldsymbol{x}_i'}{\partial t}$ and we know values of $\boldsymbol{x}_i$ from the basic Runge-Kutta simulation. Runge-Kutta simulation formulas are:

$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{6}(\boldsymbol{k}_1 + 2\,\boldsymbol{k}_2 + 2\,\boldsymbol{k}_3 + \boldsymbol{k}_4)\,h,\ \boldsymbol{y}_n = \boldsymbol{x}'(t_n),\ \boldsymbol{y}_0 = \boldsymbol{x}_0$

$t_{n+1} = t_n + h$

$\boldsymbol{k}_1 = \boldsymbol{g}(t_n, \boldsymbol{y}_n),\ \boldsymbol{k}_2 = \boldsymbol{g}\Big(t_n + \frac{h}{2}, \boldsymbol{y}_n + h\,\frac{\boldsymbol{k}_1}{2}\Big), \boldsymbol{k}_3 = \boldsymbol{g}\Big(t_n + \frac{h}{2}, \boldsymbol{y}_n + h\,\frac{\boldsymbol{k}_2}{2}\Big), \boldsymbol{k}_4 = \boldsymbol{g}(t_n + h, \boldsymbol{y}_n + h\boldsymbol{k}_3)$

Now the function $\boldsymbol{g}(t, \boldsymbol{x})$ is for gradient integration

$$\boldsymbol{g}(t, \boldsymbol{x}) = \frac{\partial \boldsymbol{x}_i'}{\partial t} = (\boldsymbol{x}(t) - \boldsymbol{x}_h(t))^T \nabla_{\boldsymbol{w}}\, \boldsymbol{f}(t, \boldsymbol{x}(t)) = \boldsymbol{\Delta}(t_i + \delta)^T\, \nabla_{\boldsymbol{w}}\, \boldsymbol{f}(t, \boldsymbol{x}(t_i + \delta))$$

But we cannot compute the modifed t_i values for the error term $\boldsymbol{\Delta}(t_i) = (\boldsymbol{x}(t_i) - \boldsymbol{x}_h(t_i))$. To work around this we need to approximate values for $t = t_i + \delta$ which are used in Runge-Kutta updates. We do linear approximation and have values for t_i and t_{i+1} and intepolate:

$\boldsymbol{\Delta}(t_i + \delta) = \boldsymbol{\Delta}(t_i) + \frac{\delta - t_i}{t_{i+1} - t_i}(\boldsymbol{\Delta}(t_{i+1}) - \boldsymbol{\Delta}(t_i))$

$\boldsymbol{x}(t_i + \delta) = \boldsymbol{x}(t_i) + \frac{\delta - t_i}{t_{i+1} - t_i}(\boldsymbol{x}(t_{+1}) - \boldsymbol{x}(t_i))$

NOTE: For the last time step we just use value $\boldsymbol{\Delta}(t_N + \delta) = \boldsymbol{\Delta}(t_N)$ and don't interpolate.

2 Independent Component Analysis and PCA

To scale to high dimensions and to find easily predictable signals from stock market prices, PCA is used to reduce number of dimensions to $O(10^2)$ signals. Signals are furher processed using ICA to find signals that don't have much higher-order correlations. The maximum variance signals are used and rest of the signals are dropped. To compute inverse of the linear preprocessing transform, pseudoinverse of linear matrix is computed.

ICA learning of 41 days signals show only static prices with no variance in data except unpredictable impulse functions (one per signal/day). This means there is no much predicatability in data except that you can divide stocks to correlating groups with similar price levels or something.

TODO: Test ICA with 6 months (120 days/samples) data.

3 Implementation

HMC sampler for learning weights $\boldsymbol{w}$ from noisy data is implemented in *Dinrhiw2* library. The code is in `src/neuralnetwork/DiffEQ_HMC.cpp`. In practice, we divide historical data to $D \times T$-dimensional vector. Vector has D values per observation and have T time steps for each values simulated by differential equations. The first D observations are initial starting point $\boldsymbol{x}_0$.

3.1 Testing

Code seem to function **quite badly** and requires more testing and enabling of bayesian prior for neural network weights which disabling did improve results.

- write testcases learning simple cases and plot the results.

*linear problem x(t)=a*t is learnt correctly, x(t)=sin(f*t) function is **NOT** learnt, neural network diff.eq. source x(t)=neural_network(x(t)) is **NOT** learn correctly but error is reduced a bit. Using same neural network as a source and starting point for training.*

TODO: write bayesian_nnetwork creating code and calculation of predictions using bayesian prediction. MAYBE: bayesian treatment can plot N randomly chosen diff.eq. models and plot the curves (no useful results).

=> DON'T WORK NOW EXCEPT IN VERY SIMPLE FUNCTIONS (LINEAR CURVES).