Summary Documentation: On-line Modeling & Fast Prediction of Time Series with Neural Networks

Goals: Finding motifs (tiny patterns), phase, amplitude patterns in time-series.
Univariate series: e.g. latency, to warn for potential fall-off
Multivariate series: seriesA (CPU) + seriesB (ethernet traffic), event/motif is hidden in the combination of motifs in two or more series).

After modeling these motifs, i.e. once model is trained, it can be used for prediction and anomaly detection.

Prototype: Since we have no data, it seems reasonable to build a prototype (black box) like grok using Machine Learning methods: starting with neural nets. Given an on-line training input, the black box provides fast predictions.

Sub-goals:

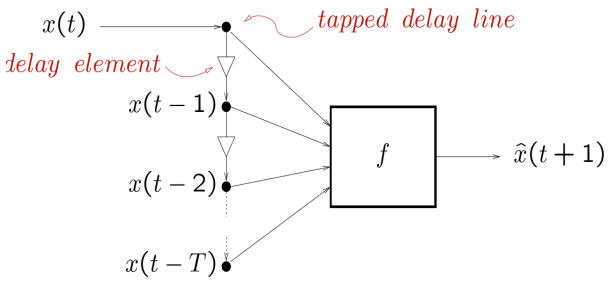
- make an python engine that learns a multi-scale neural network
- make it run stochastic gradient in real time
- predict in real time
- visualize predictions

Neural Nets: Beside any neural inspired, intelligence perks, reasons for using neural networks, as compared to graphical models are they are fast in prediction, and easy to train with off-the-shelf numerical optimizers. Also, neural networks generalizes to many tasks like classification & regression.

http://en.wikipedia.org/wiki/Feedforward_neural_network

Performance: Compared to Grok or even other methods, the first method to try out is supervised Neural Networks given lots of data. Since in theory we can generate lots of time series data, and the learning objective here is quite certain, a good optimization algorithm (online) used with neural networks will be a good bet.

On Time Series: Neural nets can be used on time-series in this manner: given the past T time steps, build a feed-forward neural net to predict the value at T+1. Since ‘f’ built by the neural net can be very complex with large number of parameters, complex patterns can be modeled by this predictor.



To predict a series of time steps, i.e. a minute of latency data at 10ms time-steps, run the feedforward model N times recursively. In this manner, it is possible to predict any desired future horizon. We would expect better performance on near term.

Recurrent Neural networks: There is further motivation to build recurrent neural networks. In theory the recurrent NNs are more powerful since they model infinite time horizon (equivalent to IIR filters), however, in practice I would guess multi-scale (multiple time lengths combined in an ensemble) feedforward will work better and a lot easier to build.

[Technical Notes] Neural Network On-line Learning Module (NOLM)

The neural network module aims to build a machine learning algorithm to perform prediction and anomaly detection in time-series. This module is able to learn from an online feed of multivariate time-series, and offer predictions over arbitrary horizon (with accuracy trade-offs) when prompted.

GitHub [Tested with Phil, he will be able to give you a demo]

https://github.com/apcera/nn_pred.git
[Run] nn_pred/dev/python/test/testreg.py

I still have interest to continue working on this repo. Please feel free to ping me if you have suggestions, or would like anything changed/explained.

Requirements:

- Python 2.7.3
- Numpy
- Scipy
- Matplotlib

Mac Superpack contains all three of the above additional packages
https://github.com/fonnesbeck/ScipySuperpack

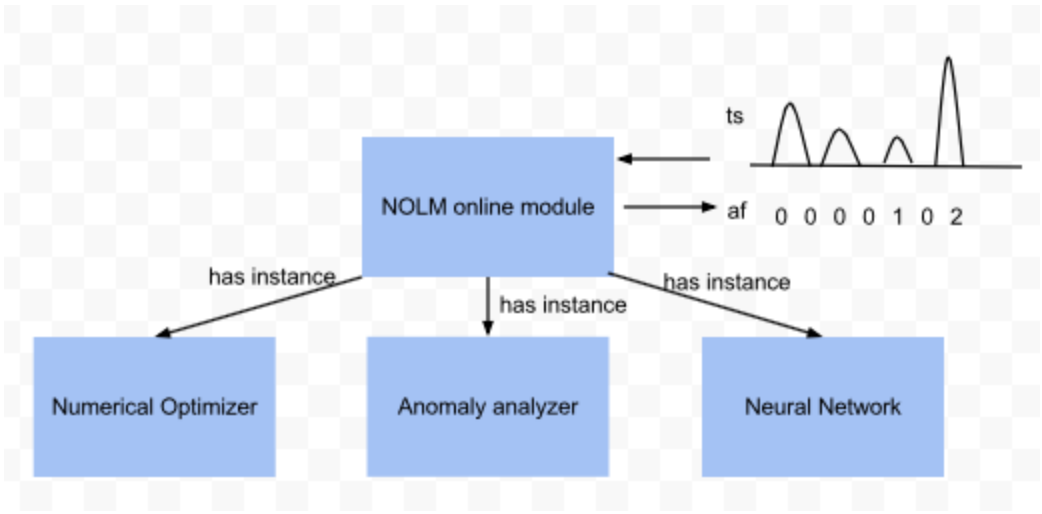
Input:

- On-line feed of multidimensional time series values x_t (CPU usage, Memory usage, traffic between nodes, latency)

Output:
- Online prediction and provide visualizations
- Send message to report (health check) an anomaly for time-series (ts_id) with flag (af), currently I assume NOLM will not take any actions (start node, shut down)

Flag types (to be researched and implemented later):
- Motif anomaly
- Significant change in event time
- Significant change in event time drift (moving towards right linearly, but sudden left)
- Significant change in amplitude change
- Significant change in amplitude change drift

Architecture:



Note the realized module is slightly different from this graph, but close have Numerical Optimizer module (nnstreamtrainer) and Neural Network (net.py). The Anomaly analyzer is not implemented and may require further research.

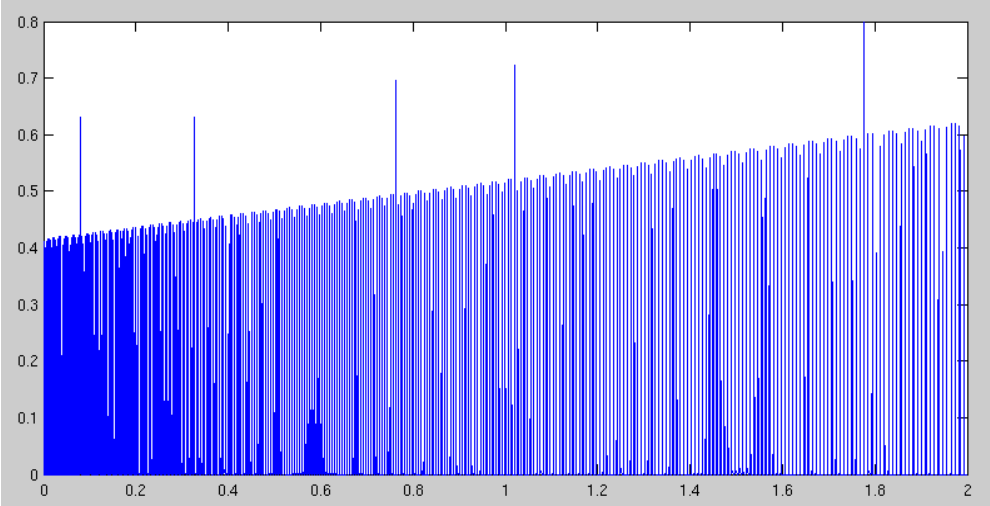
Computation Resources:
Depending on number of dimensions in the input time series, and the model complexity (analysis in progress to determine acceptable accuracy trade-offs), on-line training of NOLM may require significant CPU resources. A rough estimate is a few machines (3-5), and this is to be updated.

[Further research]

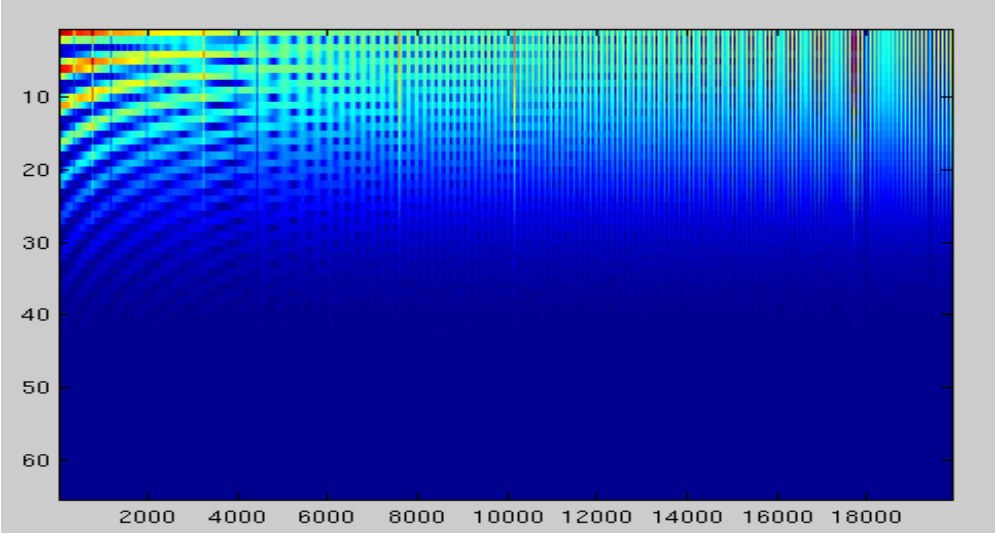
Detecting amplitude patterns:
My thoughts are detecting amplitude patterns are to track a streaming ‘mean’, further, build a temporal pyramid of subsampling sections, and use a regression model to predict the next ‘amplitude’. At prediction time this ‘mean’ is added back in the predictions to form the action mean + motif waveform.

Detecting phase patterns:

I think the solution for phase patterns is to use spectrograms. For instance the spectrograms for the following signal:



Is this patterned image:



When a neural network is used with spectrogram data, hopefully it will be able to recognize the changes in the spectrum (vertical) over time (horizontal). When it is used to predict the next data-point, the hope is it will realize ‘things should be more far apart now’ due to lower values in the high frequency ranges in the spectrum.