# **Forecasting Short Time Series with LSTM Neural Networks**

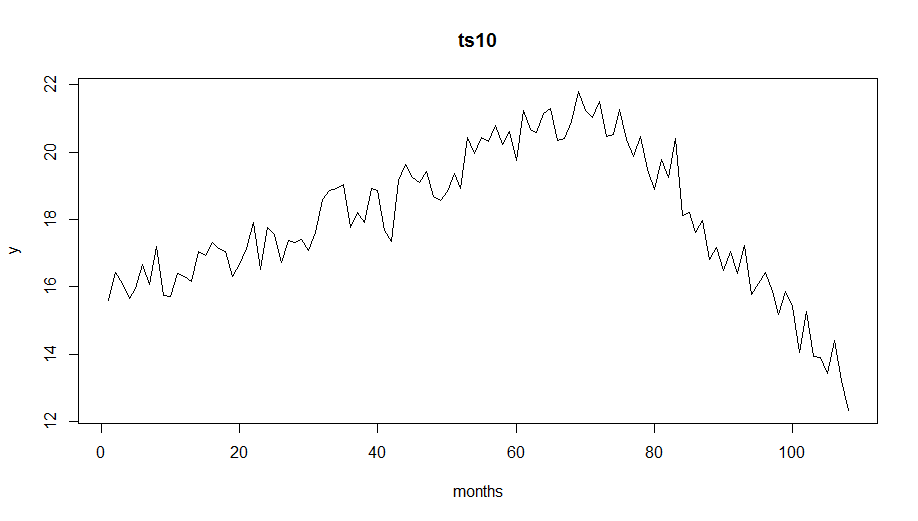
This tutorial demonstrates a way to forecast a group of short time series with a type of a recurrent neural network called Long Short-Term memory ([LSTM](http://colah.github.io/posts/2015-08-Understanding-LSTMs)), using Microsoft’s open source Computational Network Toolkit ([CNTK](https://github.com/Microsoft/CNTK)). In business, time series are often related, e.g. when considering product sales in regions.

The author used this approach to win Computational Intelligence in Forecasting (CIF) International Time Series Competition 2016 (<http://irafm.osu.cz/cif>).

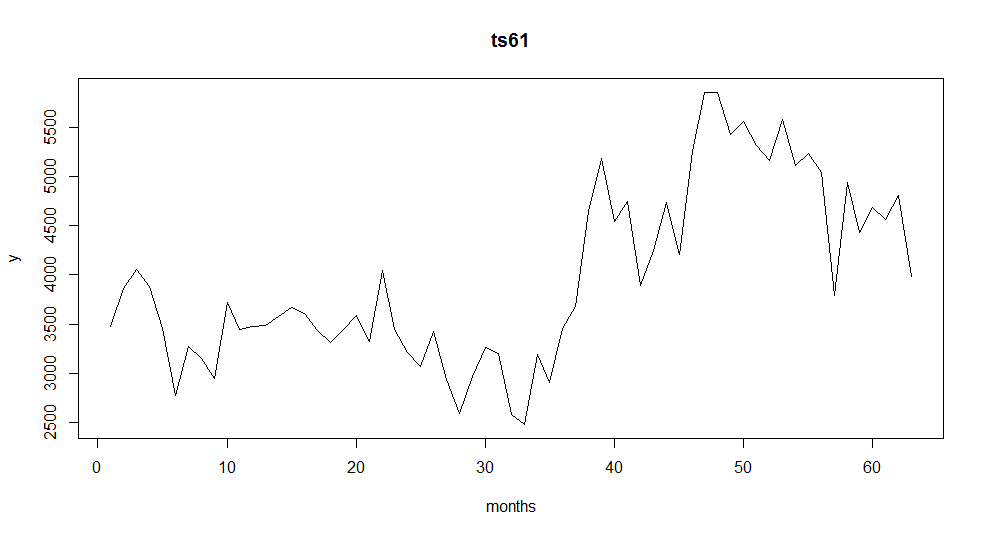
# The dataset

In the CIF 2016 competition, there were 72 monthly time series, of relatively short length (up to 108-long); 24 of them bank risk analysis indicators, and 48 were generated. In majority of cases, the contestants were asked to forecast 12 future monthly values (so, up to 1 year ahead), but for some shorter series the forecasting horizon was smaller, 6.

An example of a generated series:



And an example of a “real-life” series:



As a rule, the real time series were more volatile.

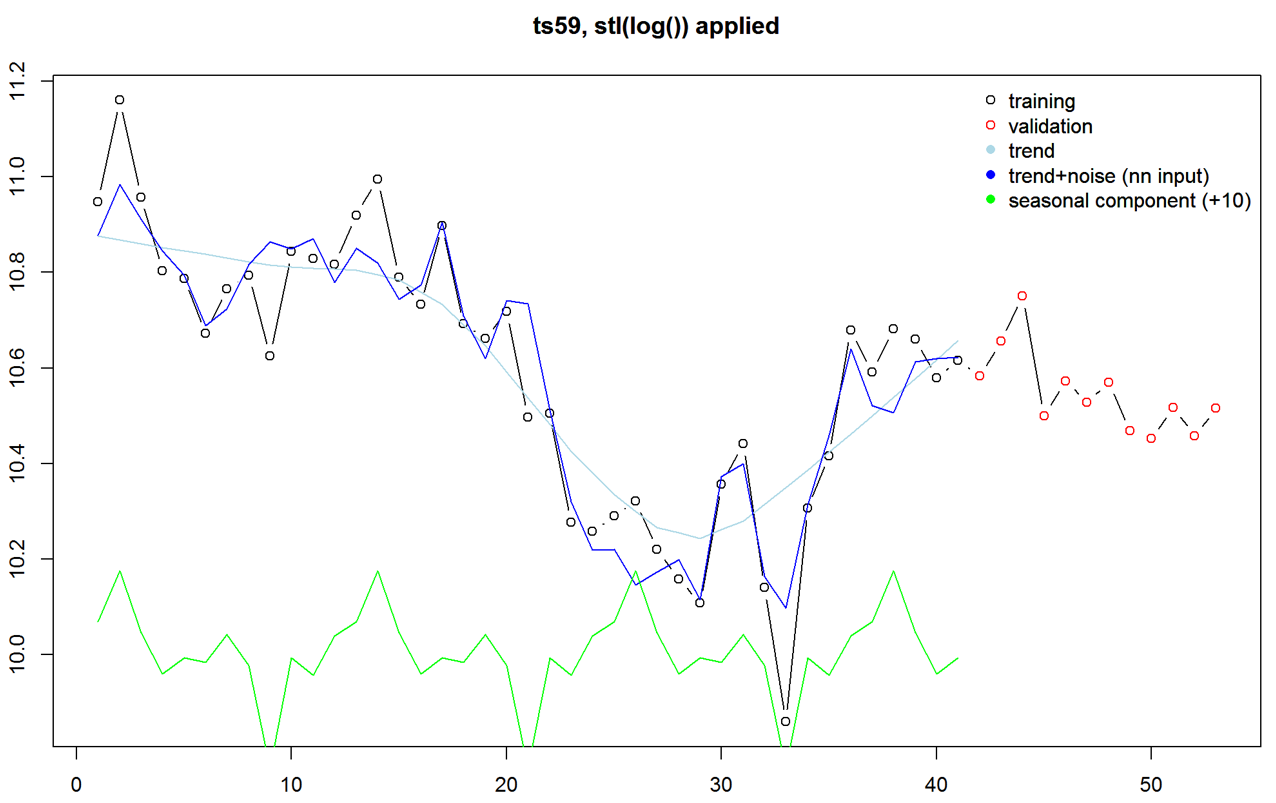
# Forecasting time series with neural networks

Neural networks have the ability to learn mapping from inputs to outputs in broad range of situations, and therefore, with proper data preprocessing, can also be used for time series forecasting. However, as a rule, they use a lot of parameters, and a single short time series does not provide enough data for the successful training. This problem can be alleviated by learning across many time series, but if using standard (non-recurrent) neural networks, this may not be a good strategy – the series may diverge a lot for similar past values. However, Recurrent Neural Networks (RNNs) have an internal state, and may learn to respond (forecast) differently series with similar short-term histories, but with dissimilar long-term histories. One of the most attractive RNNs with good long-term memory is Long Short-Term Memory network. An LSTM can be viewed as a powerful and complicated, nonetheless a single layer neural network, see a great introduction at <http://colah.github.io/posts/2015-08-Understanding-LSTMs>.

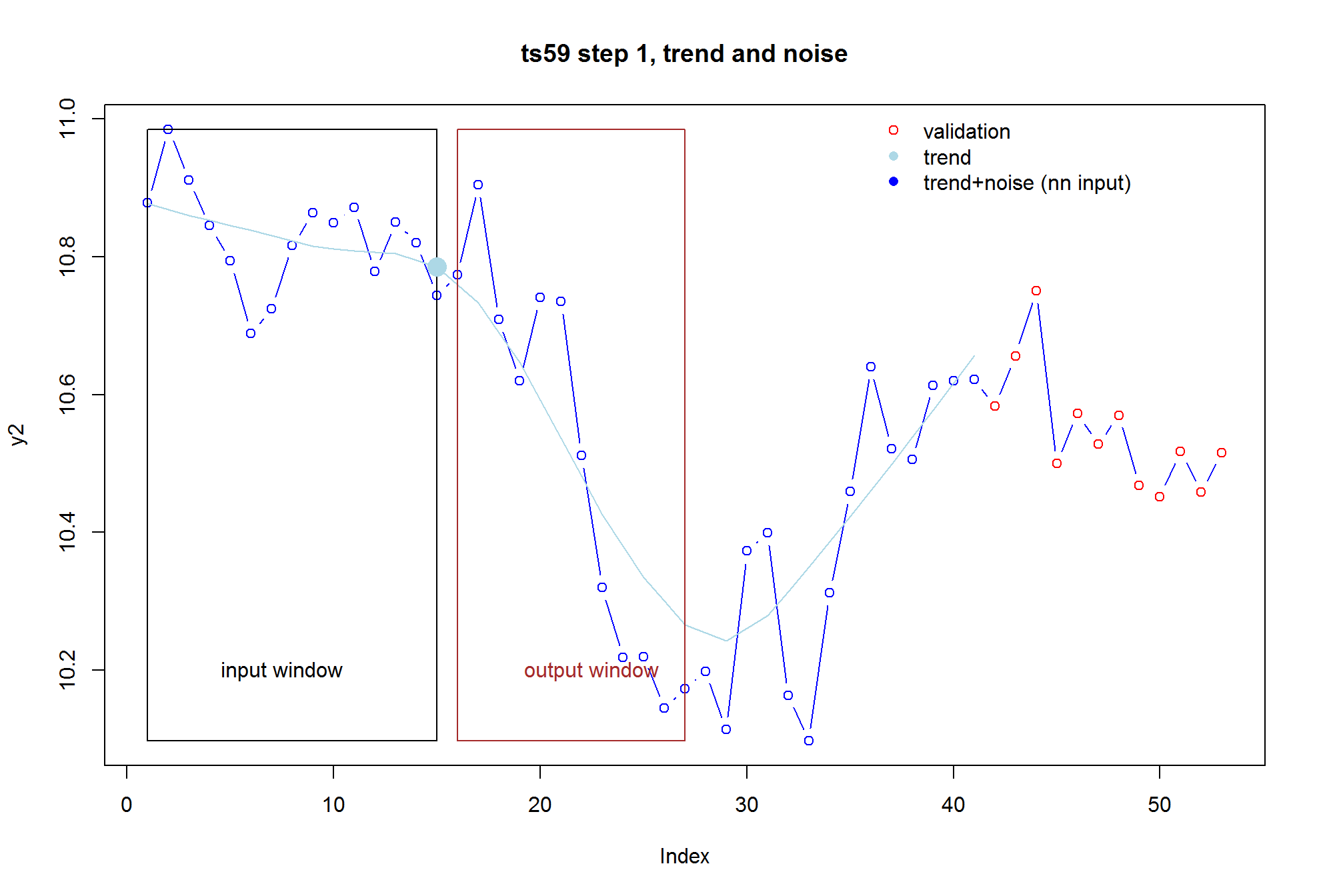
# Data preprocessing

Forecasting time series with Machine Learning algorithms or Neural Networks requires a data preprocessing. This is typically done with a moving (or “rolling”) window along the time axis; at each step, constant size features (inputs) and outputs are extracted, and therefore each series will be a source of many input/output records. For networks with squashing functions like sigmoid, a normalization is often helpful, and it is more needed if we try to train a single network of this kind on many time series differing in size. Finally, for seasonal time series, although in theory a neural network should be able to deal with them well, it often pays off to remove the seasonality before applying a neural network.

Have a look at [**cifPrepStl.R**](https://microsoft-my.sharepoint.com/personal/slsmyl_microsoft_com/Documents/docs/cifTutorial/src/cifPrepStl.R). It downloads and preprocesses the competition data set producing 4 files: training and validation, separately for time series with 6 and 12-long forecasting horizons. It starts with applying logarithm and then the stl() functions of R. STL decomposes a time series into seasonal, trend, and irregular components. The logarithm transformation provides two benefits: firstly, it is part of the normalization, secondly, it converts the STL’s normally additive split into the multiplicative one (remember that log(x+y)=log(x)\*log(y)), and the multiplicative seasonality is a safer assumption for non-stationary time series. A graph below illustrates this decomposition:



So, after subtracting the seasonality, the moving window was applied covering 15 months (points) in case of 12 months’ ahead forecast. It is perhaps worth stressing: the input is 15-long vector and the output is 12-long vector, so we are forecasting the whole year ahead at once. This approach works better than trying to forecast just one month ahead – in such a case to get the required 12-step (month) ahead forecast we would need to be done 12 times, and use previous forecast as input 11 times. That leads to instability of the forecast.



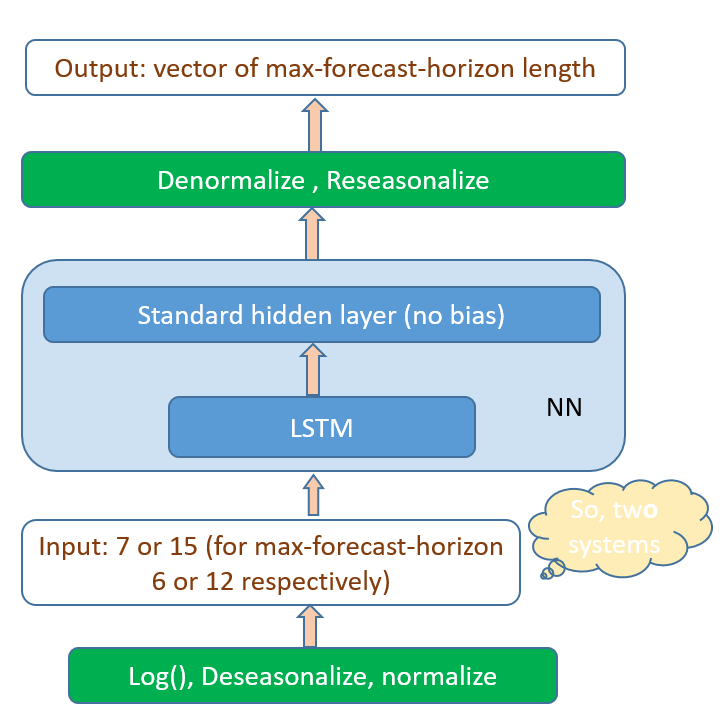
Then, the last value of *trend* inside the input window (the big filled dot above) is subtracted from all input and output values for normalization. Input and output windows move forward one step and the normalization step is repeated. Two files are created: training and validation. The procedure described above continues until the last point of input window is positioned at lengthOfSeries-outputSize-1, e.g. here 53-12, in case of the training file, or until the last point of the output window equals the last point of the series, in case of the validation file. The validation file contains the training file, but actually only the last record of each series is used – the rest, although later forecasted, is discarded and only used as a “warm-up” region for the recurrent neural network used (LSTM).

The data preprocessing described here is relatively simple, it could have been more sophisticated, by including some other features of time series.

Run cifPrepStl.R, from command line by executing e.g. C:\Program Files\R\R-3.3.1\bin\rscript cifPrepStl.R (so full path to the rscript.exe followed by the script name) or, if you have RStudio installed, by opening supplied **cif.Rproj**,and running cifPrepStl.R within RStudio.

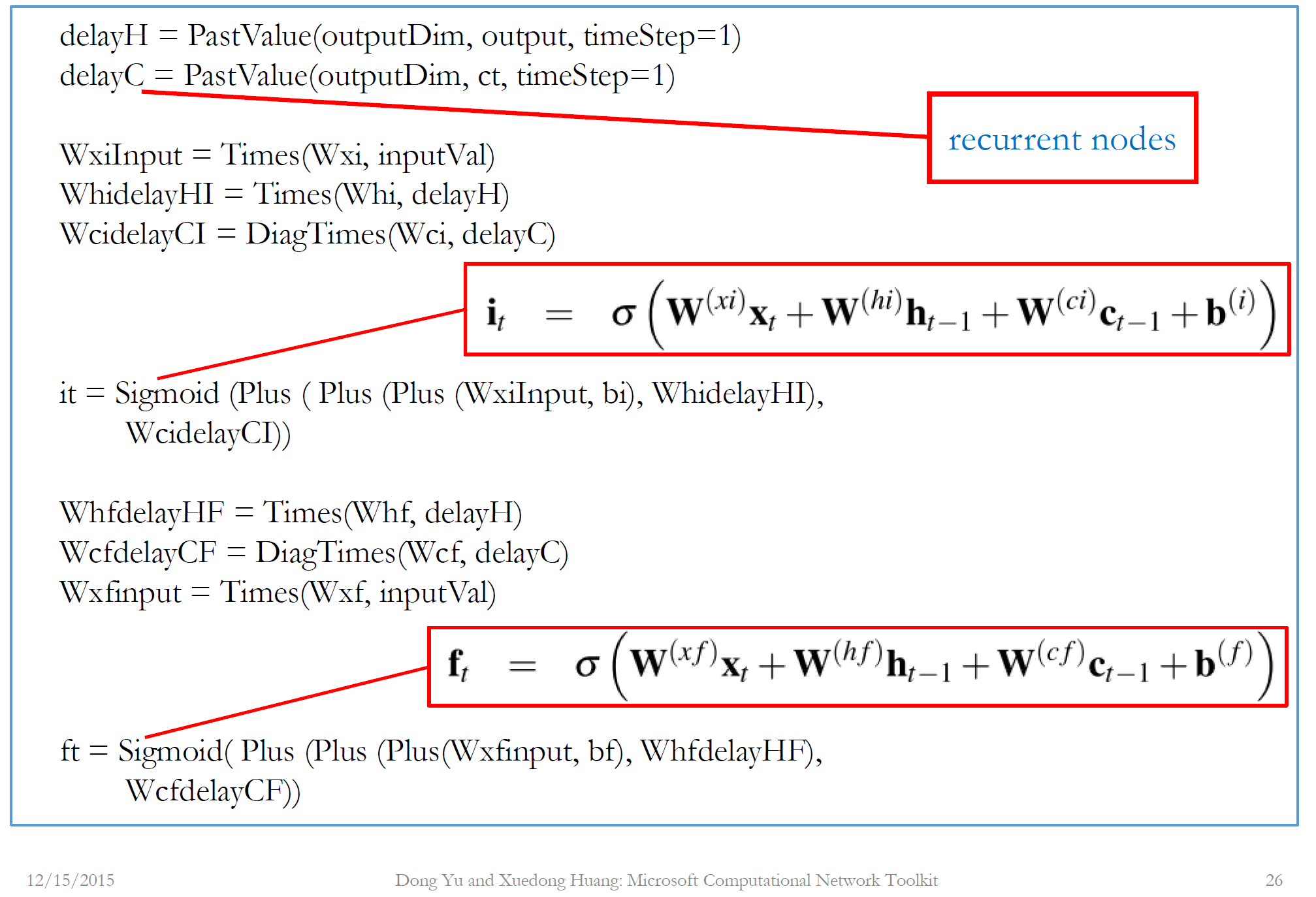
# The system architecture

The neural network consists of a single LSTM network and a simple linear “adapter” layer without bias. The whole solution is pictured below:



# Computational Neural Toolkit and its configuration file

I used Computational Neural Toolkit (CNTK). It is Microsoft’s open-source neural network toolkit, available for Windows and Linux. It is very scalable – it can run on a CPU, one or more GPUs in a single computer, but also on a cluster of servers, each running several GPUs. This scalability was not actually needed for this project – the whole learning was taking just a few minutes on a PC with a single GPU. More important was the ease of creation and experimentation with neural networks architectures. CNTK shines here – it allows to express almost arbitrary architectures, including recurrent ones, through mathematical formulas describing feed-forward flow of the signal. For example, the figure below shows beginning of definition of a LSTM network; note how easily is to get a past value for a recurrent network, and how straightforward is translation from the mathematical formulas to the code.



Actually, you will not find any of these formulas in the CNTK’s script **cifStl\_bs17.config**. It is because I rewrote my original script that included them, into a new one (for the current, as of September 2016, version 1.7) that utilizes preconfigured LSTM network. Have a look inside. Instead of two pages of formulas, there are two lines in *BrainScriptNetworkBuilder* section. They specify the model as being a sequence of two layers, *RecurrentLSTMLayer* and a *DenseLayer* (this is the adapter layer without bias). Then inputs and outputs are declared, and the learning loss function (*SquareError*) specified.

In the *SGD* section, some learning parameters are specified including epoch size, number of epochs, and the amount of noise added to the weights gradient (a useful technique that reduces chances of overfitting). Typically, the epoch size is 0, meaning that all training examples make up one epoch, but here, because the training set is relatively small (a bit over 3k) I changed it to 10k, so not to be overwhelmed by the info CNTK outputs to the console after every epoch.

After the training is done the *nn\_Write* command is executed. It writes out values of the output node (the forecast vector). Please note that in the *reader* section, *randomize* is set to false, unlike in nn\_Train command where it is true to facilitate learning. It is done to ensure that the sequence of records in the output file will match exactly the sequence in the validation file, crucial for the next step, postprocessing.

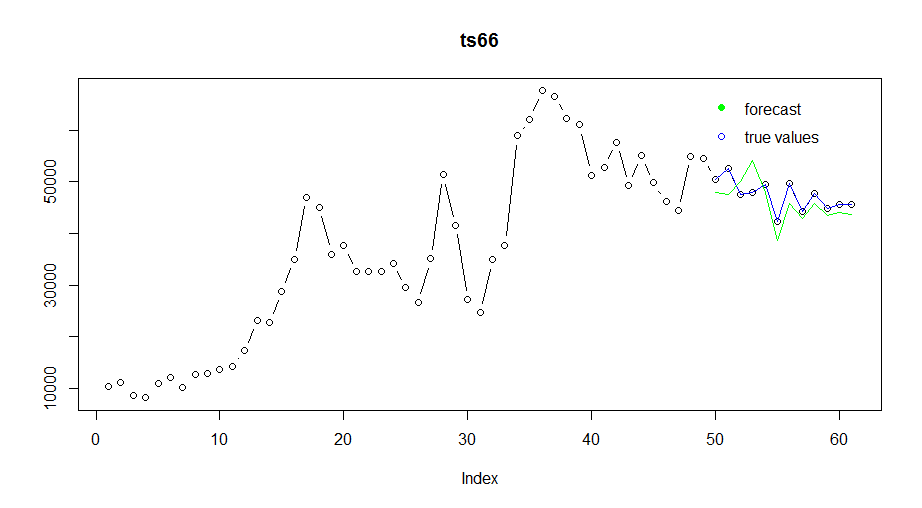
You run the CNTK script in following way. Open cmd prompt (or a Linux terminal), go to the directory where you downloaded the code, and, assuming that you have already run cifPrepStl.R script and the prepared data in in *data* subdirectory, execute

<path to cntk subdir of the CNTK distribution>/**cntk configFile=cifStl\_bs17.config**

# Postprocessing

The purpose of **cifValidStl.R** script is to calculate average metrics: bias and sMAPE (for the definition of the latter see <https://en.wikipedia.org/wiki/Symmetric_mean_absolute_percentage_error>, it was the metric used by the CIF 2016 competition) of the forecast on the last record of each of the time series, and thus guide choosing preprocessing algorithm, training architecture, parameters etc. I would like to make it clear: although the CNTK’s output file contains the same number of records as the validation file (around 5k), only last record of each time series is considered, thus we are calculating and averaging 72 values – one for each time series. But it would not be a good idea to skip all of these preceding records in the write step of CNTK – they are necessary for a recurrent network to “zero-in” on a particular time series, to establish the state.

As in case of the CNTK’s script cifStl\_bs17.config, two constants define input and output size, 15 and 12 respectively, and the script processes the majority of time series with 12-months ahead forecasting horizon. At the end, a randomly selected curve is shown with the forecast, as displayed below:



# Final steps

After all the experimentation is over, we settle on the best data preprocessing, network architecture, and learning parameters. To make the final forecasts we need a few additional steps that are very similar to the above steps, just with some differences: First we prepare new “test” file that contains all the data, we learn on the previously created validation file, and then forecast (output from neural network) using the new test file, then extract forecasts for last record of each time series. (And do the learning, outputting, and extraction twice for both 6 and 12-months ahead series).

Done ☺ Now you could go to <http://irafm.osu.cz/cif/main.php?c=Static&page=download> and download the testing data set (true values uncovered only after competition ended) and see how you have done.