Our Project Title

Table of contents I

Section 1

Abstract

Arima

This project delves into the theoretical framework of two types of models. It aims to compare and contrast the performances of the moving average model and the artificial neural network on the prediction of stock market indices. It also aims to compare how the abnormal data influence two models(pandemic, financial crisis, etc.) It compares the performances of two models on indices of different sectors and markets. We used the data collected from Yahoo Finance with daily frequency from 1 January 2000 to 31 December 2019. We use a rolling window approach to compare ARIMA with the hybrid models to determine whether hybrid ARIM-SGARCH can reflect the specific time series characteristics and predict better.

Data Analysis

We perform data analysis and a model-fitting procedure for the logarithmic returns of the S&P500 index. We get the data from Yahoo Finance. Given the hybrid ARIMA-GARCH model, we analyze 19 years of data. We transform the adjusted price into a daily logarithmic return,

$$r_t = ln \frac{P_t}{P_{t-1}}$$

Section 2

Models

ARIMA SGARCH - Overview

In practice, stock prices can be tremendously volatile during economic growth as well as recessions. In such scenarios, when homoskedasticity presumption is violated, it is said that the errors are heteroskedastic. Given that heteroskedasticity can affect the validity or power of statistical tests when using ARIMA models, we consider the ARCH effect. In this model, the error term of the ARIMA model in this process follows SGARCH(1,1) instead of being assumed constant like the ARIMA model.

Firstly, we conducted a rolling forecast based on an ARIMA model with window size(s) equal to 1000. The optimized combination of p and q which has the lowest AIC is used to predict return for the next point. At the end, the vector of forecasted values has the length of 3530 elements, with a starting point at 20 December 2005

Next, we describe and review our implementation of dynamic ARIMA(p,1,q)-SGARCH(1,1) models with GED distribution and window size(s) equal to 1000 and where optimized ARIMA(p,1,q) is taken from

ARIMA SGARCH - Overview

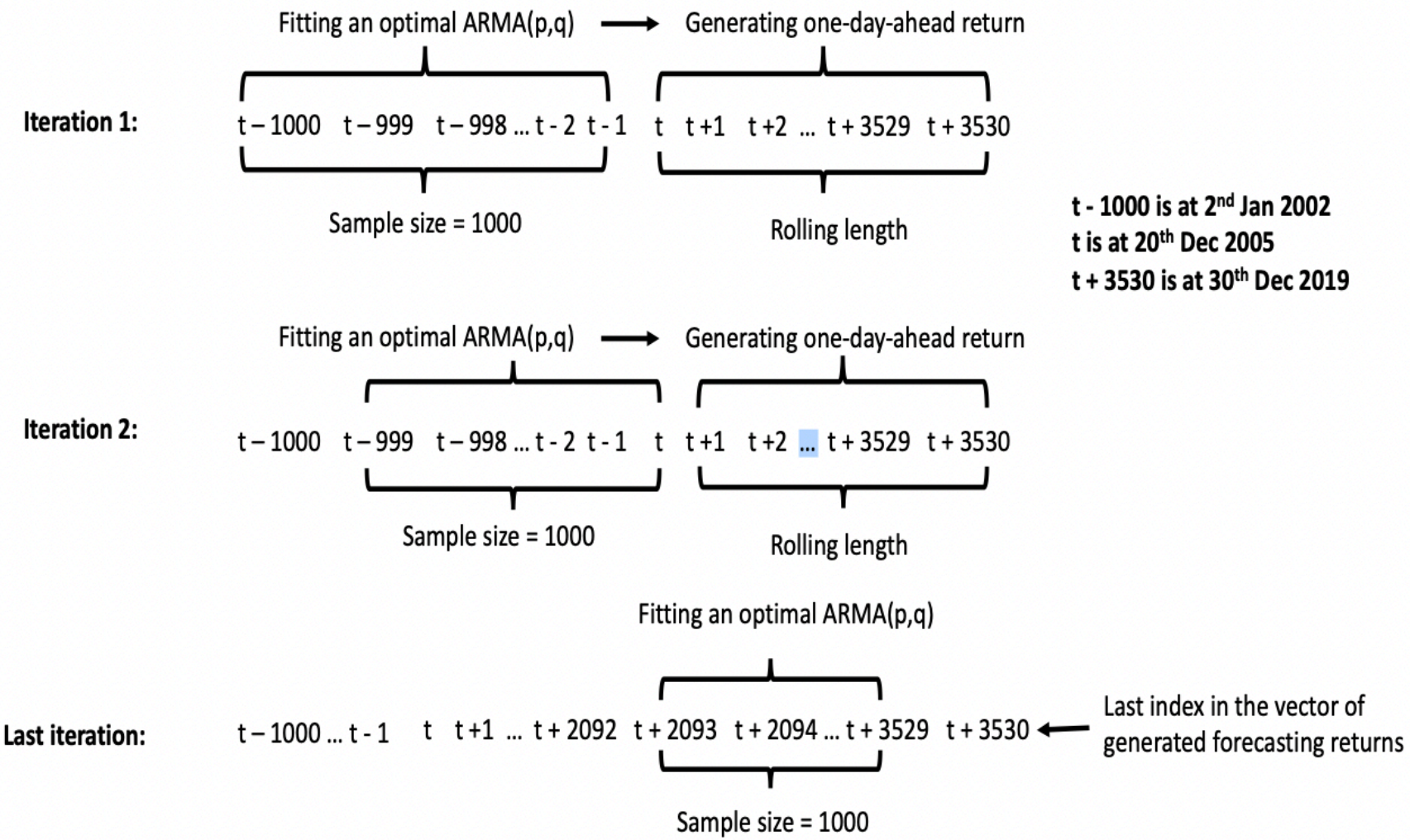
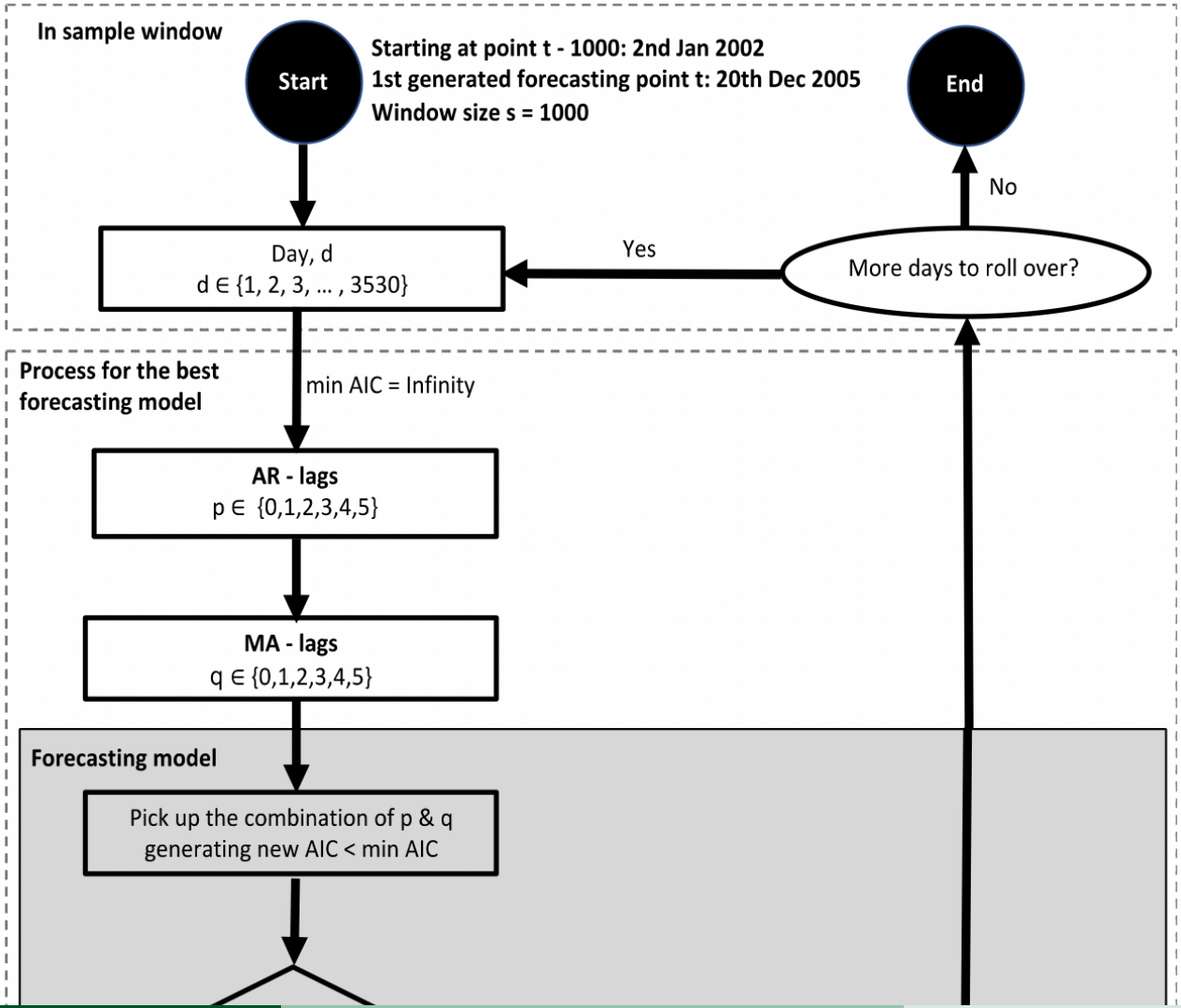


Figure 1: ARIMA mechanism

ARIMA SGARCH - Methodology

We first fit ARIMA in the following way:



Error Metrics

In order to evaluate forecast form estimated models, we calculated the following error metrics: Mean absolute error (MAE)

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|A_i - F_i|$$

where: n is the number of errors; A_i is the actual value and F_i is the forecasted value computed by the given model.

Mean square error (MSE)

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(A_i - F_i)^2$$

Section 3

RNN-LSTM Model

Recurrent Neural Network(RNN)

A **recurrent neural network** is a class of artificial neural network that uses sequential or time series data. Unlike Feedforward Neural Network, RNN allows the output from some nodes to affect subsequent input to the same nodes by using connections between nodes to create cycles. As a result, the hidden layers produce the outputs with the input information and prior "memory" received from previous learning.

Unroll RNN

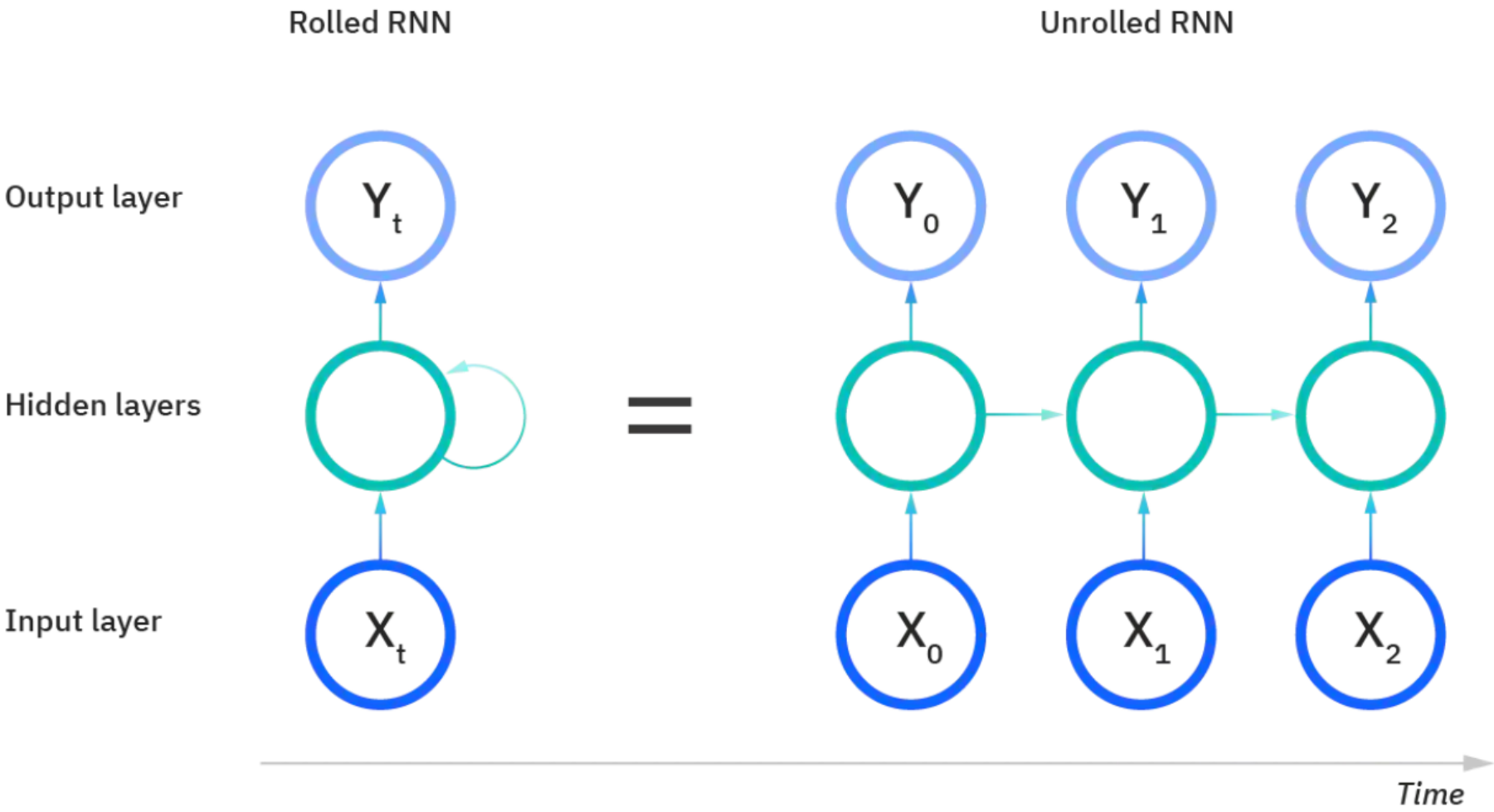


Figure 3: Rolled RNN and Unrolled RNN

Recurrent Neural Network(RNN)

Another distinguish characteristic of RNN is that they share parameters across each layer of the network. Unlike feedforward neural networks having individual weight and bias for each node in one layer, recurrent neural networks share the same weight parameter within each layer. However, these weights are still adjusted during the processes of backpropagation and gradient descent to facilitate reinforcement learning.

In feedforward neural network, backpropagation algorithm was used to calculate the gradient with respect to the weights. Recurrent neural network, on the other side, leverage backpropagation through time (BPTT) algorithm to determine the gradient as BPTT is specific to sequential data.

Activation Functions

In neural networks, an activation function determines whether a neuron should be activated and typically maps the input to $[0, 1]$ or $[-1, 1]$. The followings are two of the most commonly used activation functions and will be adopted later:

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Tanh (Hyperbolic tangent)

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU (Rectified Linear Unit) Activation Function

$$R(x) = max(0, x)$$

Long Short-term Memory (LSTM)

Long short-term memory network, usually known as LSTM, is a specific RNN architecture first introduced by Sepp Hochreiter and Juergen Schmidhuber as a solution to vanishing gradient problem. Recall with an RNN, similar with human reading a book and remembering what happened in the earlier chapter, it remembers the previous information and use it for processing the current input. The shortcoming of the NN is that it is not able to remember long term dependencies due to the vanishing gradient. The LSTM is designed to alleviate and avoid such issues.

The LSTM consists of three parts:

- **Forget Gate**: Choose whether the information coming from the previous time stamp should be remembered or can be forgotten
- **Input Gate**: Learn new information from the input to this cell
- **Output Gate**: Passes the updated information tot the next time stamp

Forget Gate

In an LSTM cell, the cell first need to decide if the information from previous time stamp should be kept or forgotten. The equation of the forget gate is:

$$f_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f)$$

Where

- x_t = input to the current time stamp
- h_{t-1} = hidden state of the previous time stamp
- W_f = weight matrix associated with hidden state
- b_f = constant

After that, a `sigmoid` function is applied over f_t and make it a number between 0 and 1. Then f_t is multiplied with the previous cell state. If $f_t = 0$, the network will forget everything from the previous time stamp while $f_t = 1$ represents that the network will remember everything.

Input Gate and new information

Next we decide what new information we will store in the cell state. First, the input gate decides which values we'll update with `sigmoid` activation function:

$$i_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i)$$

Where

- W_t = weight matrix of the input associated with hidden state

Next, the new information is sent through a `tanh` layer to create the new candidate values:

$$\tilde{C}_t = tanh(W_C \cdot [x_t, h_{t-1}] + b_C)$$

New Cell State C_t

With previous work, the LSTM cell now updates the new cell state for the current time stamp as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The current cell state C_t combines how much we decide to remember from the previous cell state C_{t-1} scaled by the forget gate and how much we wish to take in from the new current input $\tilde{C}_t$ scaled by the input gate.

Output Gate

Finally, the cell needs to decide what it is going to output and by how much. The filter of the output is the output gate, with the following equation:

$$o_t = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o)$$

The equation of the output gate is very similar with the forget gate and the input gate. Then, we push the cell state C_t through the `tanh` activation function to maintain the value staying in between -1 and 1, and multiply it by the output gate:

$$h_t = o_t * tanh(C_t)$$

Overall Module

The previous steps conclude the architecture of the LSTM. The whole process can be summarized and displayed as the following:

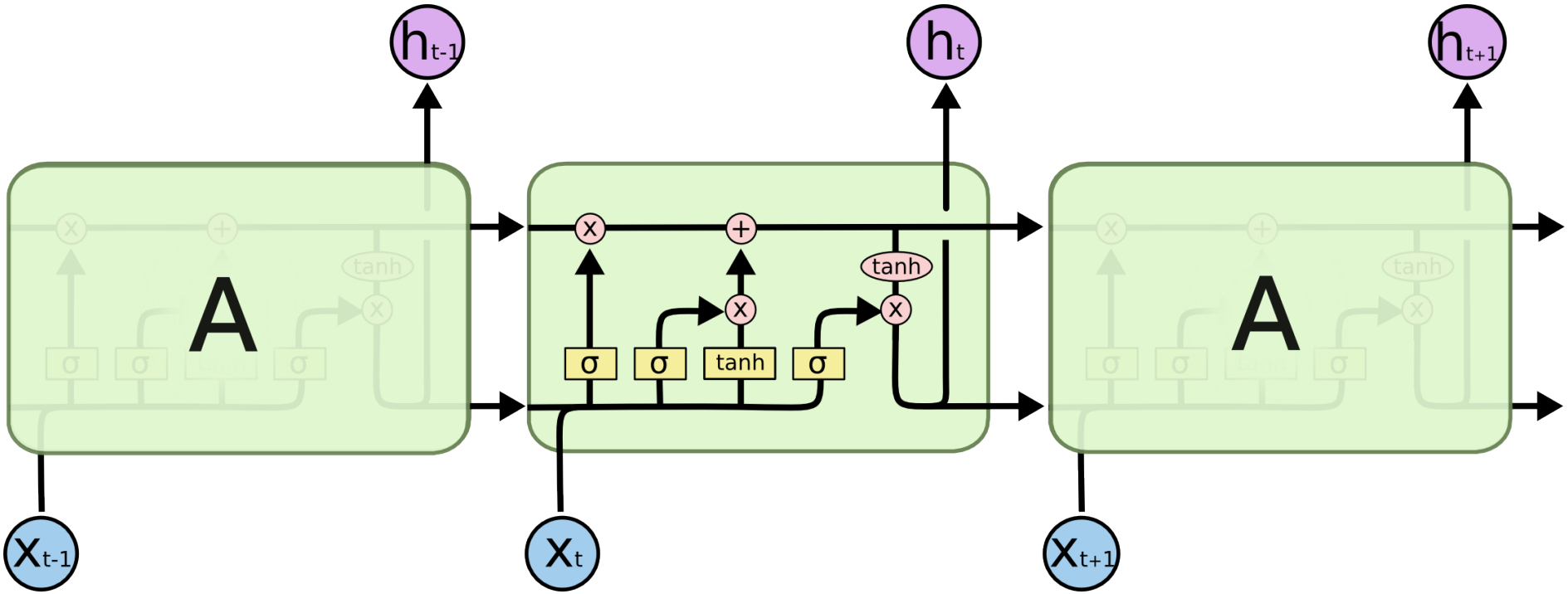


Figure 4: LSTM Chain

Application in Tensorflow

We implement the LSTM by using the package `tensorflow`. The LSTM model in the `keras` package of `tensorflow` follows our previous description by using the `tanh` function as the activation function and the `sigmoid` function as the recurrent activation function.