RETURN AND VAR PREDICTION USING RNN

Minna Zhang

May 9, 2024

1 Introduction

- **RNN** RNN processes inputs one at a time and then return output in the output layer. In the meantime, the output of intermediate layers of the previous step will be delivered to the next layer as part of the input. This ongoing recursive process continues for every element of a sequence until we obtain a prediction of the final output.
- **LSTM** Long Short-Term Memory networks usually called LSTMs are a special kind of RNN capable of learning long-term dependencies. LSTMs have a chain-like structure, but the repeating module has a slightly different structure. There are multiple layers which interact in a very special way. A common LSTM architecture is composed of a memory cell, an input gate, an output gate and a forget gate. Three of the gates can be thought of as a conventional artificial neuron, as in a multi-layer or feedforward neural network that computes using an activation function of a weighted sum.
- **Why LSTM** In deep learning, there are two factors that affect the magnitude of gradients - the weights and the activation functions, especially their derivatives, which the gradient passes through. If either of these factors is smaller than 1, then the gradients may vanish in time; if larger than 1, then exploding might happen. In the recurrency of the LSTM the activation function is an identity function with a derivative of 1.0. So, the backpropagated gradient neither vanishes nor explodes when passing through, but remains constant. The effective weight of the recurrency is equal to the forget gate activation. So, if the forget gate is on (activation close to 1.0), then the gradient does not vanish. Since the forget gate activation is never greater than 1.0, the gradient cant explode either. Thats why LSTMs are so good for learning long range dependencies and are well-suited to classify process and predict time series given time lags of unknown size and duration between important events. This is how LSTMs were developed to deal with the exploding and vanishing gradient problem when training traditional RNNs.

2 General Structure

The train-test data first needs to be identified. Then the data set is scaled using MinMaxScaler in order to train the data. A batch function is defined before setting up the RNN. There are important parameters that need to be tuned to develop the optimum result. After the RNN setup, placeholders are defined to hold the desired values, an LSTM cell is defined to deal with short term and long term memory, and dynamic RNN and loss are defined to calculate the Mean Squared Error (MSE). After evaluating the MSE the model decides whether it should run the session to get the final result for predicted price or re-run again for tuning the parameters of the RNN setup. Finally, after necessary iterations, the model is able to predict the stock returns very precisely and accurately.

- **Overview:** Brief introduction to the importance of accurate modeling of market returns and estimating financial risks, specifically Value-at-Risk (VaR).
- **Objective:** Explain the objective of using Long Short-Term Memory (LSTM) networks to enhance the predictions of market return parameters and VaR estimation.
- **Significance:** Highlight the significance of LSTMs due to their ability to capture long-term dependencies in financial time series data.

- **Model Structure** The LSTM model for market returns would typically involve input layers that take sequences of past return data (and possibly other financial indicators), hidden LSTM layers to process this data, and output layers that predict the necessary parameters (mean and volatility).
- **Data Preparation** The data fed into LSTMs for this purpose would be sequences of returns, possibly normalized or standardized to improve model training efficiency.
- **Training the Model** The LSTM model would be trained on historical data, learning to predict the next time step's mean and volatility based on the sequence of past data. This training involves backpropagation through time and can be optimized using techniques suitable for time series data.
- **Application** The trained LSTM model would be used to generate predictions for the mean and volatility of returns. These predictions can then be used in the formula for VaR calculation, combining the predicted mean and volatility with the quantile of the distribution corresponding to the desired confidence level.

3 Implementing RNNs for P&L Predictions

3.1 Objective

To use RNNs for predicting the Profit and Loss (P&L) from selected stock portfolios by analyzing historical data patterns and stock price movements.

3.2 Data Collection

The first step involves gathering historical price data and other relevant financial metrics (like trading volume, market capitalization, and macroeconomic indicators) for selected stocks. This data might come from financial markets databases like Yahoo Finance, Google Finance, or financial institutions that provide historical stock data.

3.3 Data Preprocessing

Before feeding the data into the RNN, it undergoes several preprocessing steps:

- Normalization: Scaling the data to a common scale without distorting differences in the ranges of values to help the neural network converge more quickly.
- Sequencing: Creating sequences of past stock data (e.g., closing prices for the past 60 days) to predict the next day's P&L.

3.4 RNN Model Design

Choose an RNN architecture, typically LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Units), because of their ability to capture temporal dependencies and avoid the vanishing gradient problem common in standard RNNs. (We choose LSTM here)

- Input Layer: Feeds the sequence data into the model.
- Hidden Layers: Multiple LSTM layers can be utilized to learn the data's high-level features.
- Output Layer: A dense layer with one neuron to output the predicted P&L for the next day.

3.5 Training the Model

The RNN is trained using a backpropagation algorithm over the historical data, adjusting the model weights based on prediction errors, optimizing perhaps through Mean Squared Error (MSE) or another loss function that quantifies the difference between predicted P&L and actual P&L.

3.6 Model Implementation and Prediction

Once trained, the RNN model can predict the daily P&L for the stock portfolio. By running the trained RNN with the most recent data, it can generate P&L forecasts that help financial analysts and portfolio managers in decision-making processes.

4 VaR Calculation Using RNN Predictions

4.1 Objective

To calculate Value-at-Risk (VaR) using stock price forecasts from RNNs to estimate potential losses under normal market conditions over a set time period (e.g., daily VaR).

4.2 Integrating RNN Forecasts

With the stock price predictions obtained from the RNN, simulate the potential future values of a stock portfolio over the desired time horizon. This could involve generating multiple forward-looking paths (Monte Carlo simulations) based on the volatility and price trends learned by the RNN.

4.3 VaR Calculation Method

- Parametric Approach (Variance-Covariance): Calculate the standard deviation (volatility) of the RNN-predicted prices and apply the formula for VaR assuming normally distributed returns. For a 95% confidence level, the VaR would be calculated as the mean predicted loss plus 1.65 (z-value from standard normal distribution) times the standard deviation of the losses.

5 Analysis and Reporting

The calculated VaR figures give a quantified estimate of maximum expected losses under normal market conditions, which is crucial for risk management. This model allows risk managers to see how predicted changes in stock prices could impact the portfolio, providing a proactive tool for managing financial risk.

6 Essay Structure

Step 1: Data Collection

- **What It Means:** Gather historical data about the stock market, which includes stock prices, trading volumes, and other financial details over time.
- **How It's Done:** Obtain this data from financial websites, databases like Yahoo Finance or Google Finance, or directly from stock exchanges.

Step 2: Data Processing

- **Normalization:** Adjust numbers to make them comparable, scaling prices between 0 and 1.
- **Sequencing:** Arrange data into sequences that the LSTM can learn from, similar to organizing events for a narrative.
- **Split to Train-Test Data**

Step 3: LSTM Network Components

An LSTM (Long Short-Term Memory) network is a type of RNN (Recurrent Neural Network) suited for handling sequences and time-series data. It has a unique structure that helps it remember long-term dependencies and avoid issues like the vanishing gradient problem.
The LSTM-cell consists of a forget gate, an input gate, and an output gate. The main purpose of the LSTM-cell is to remember information over time intervals while the gates control the information that flows in and out of the cell. Essentially, the gates compute which information that is to be remembered and which information that is to be forgotten. The information that is remembered in the LSTM-cell is stored in its cell state C_t and is calculated in accordance with the input x_t, the computation in the forget gate f_t as well as the previous output h_{t-1}.

Forget Gate:

- The forget gate decides what information should be discarded from the cell state. It looks at the previous hidden state (h_{t-1}) and the current input (x_t), and outputs a number between 0 and 1 for each number in the cell state (C_{t-1}).
- **Formula:** $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Input Gate:

- The input gate decides which new information is going to be stored in the cell state. It creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
- **Formula:** $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

Cell State Update:

- The cell state, C_t, is updated by forgetting the things decided to forget earlier and then adding new candidate values scaled by how much we decided to update each state value.
- **Formula:** $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Output Gate:

- The output gate decides what the next hidden state, h_t, should be. The hidden state contains information about previous inputs. The contents of the cell state are usually filtered by the output gate to decide what should be passed to the output.
- **Formula:** $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
- $h_t = o_t * \tanh(C_t)$

Step 4: Training the LSTM

This step focuses on optimizing the LSTM network by adjusting its weights through a process known as Backpropagation Through Time (BPTT). This involves:

- Using a loss function (commonly Mean Squared Error) to evaluate the accuracy of predictions, quantifying the difference between actual values and the model's predictions.
- Applying an optimizer (like Adam or SGD) to minimize the loss by iteratively adjusting the weights of the network.

Training the model effectively ensures that it learns to generalize well from the historical data it has been exposed to, which is essential for reliable future predictions.

- **Backpropagation Through Time (BPTT):** Update network weights by calculating gradients of the loss function.
- **Loss Function:** Typically Mean Squared Error (MSE) for regression tasks.
- **Formula:** $MSE = \frac{1}{n}\sum(y_{\text{actual}} - y_{\text{predicted}})^2$
- **Optimizer:** Methods like Adam or SGD to adjust weights based on gradients.

Step 5: Making Predictions

Once the LSTM is trained, this step uses the model to make forward-looking predictions based on new data sequences. This is the direct application of the LSTM for forecasting, using the learned patterns to predict future values of, for example, stock prices or market volatility.

- **Feedforward Through LSTM:** Process new data sequences through the LSTM to generate predictions.

Step 6: Preventing Overfitting in LSTM Models

ONLY DO THIS PART IF WE HAVE TIME
Overfitting is a critical issue where a model learns the details and noise in the training data to an extent that it negatively impacts the performance of the model on new data. This step involves techniques such as:

- **Early Stopping:** To avoid overfitting, implement the early-stopping method, which halts the training process when the loss on the validation set no longer shows improvement, thus preserving the model's ability to generalize.
- **Dropout:** Utilize dropout techniques that randomly ignore selected neurons during the training process, effectively thinning the network temporarily and reducing the risk of overfitting. This randomness helps prevent co-adaptations on training data.

Step 7: Application of LSTM to Estimate μ_t and σ_t

- **Modeling μ_t and σ_t:** Define μ_t and σ_t as outputs from separate LSTM networks represented by the functions:

$$\mu_t = f_1(\Omega_{t-1}; \theta)$$
$$\sigma_t^2 = f_2(\Omega_{t-1}; \theta)$$

- **Training Details:** Describe the data preparation, normalization process, and LSTM training specifics (e.g., learning rates, epochs).

We denote Ω_{t-1} as the set of information available at time t-1 and θ as the set of parameters, then under this framework the conditional mean t and conditional volatility t can be estimated by two nonlinear functions $f_1(\Omega_{t-1}; \theta)$ and $f_2(\Omega_{t-1}; \theta)$ which can be approximated by two RNN trained separately.

Step 8: Market Returns Formula

This approach can be used daily by financial institutions to assess their risk exposure and adjust their investment strategies accordingly, ensuring they are prepared for potential adverse movements in the market.

- **Market Returns:** Define market returns with a focus on log returns as defined by the equation:

$$r_t = \mu_t + z_t\sigma_t$$

where:

- r_t: log return at time t
- μ_t: conditional expected mean of log return
- σ_t: conditional volatility of log return
- z_t: innovation variable with zero mean and unit standard deviation

Step 9: Parametric Approach (Variance-Covariance)

The Parametric Approach, also known as the Variance-Covariance method, is a way to calculate VaR based on assumptions that market returns are normally distributed. Here's a breakdown of the formula and each term:

- **Formula:** $VaR_t(\alpha) = \mu_t + z_\alpha \cdot \sigma_t$
- **Variables:**
  - μ_t is the predicted mean (average) return at time t. This is the average outcome expected from an investment based on historical data.
  - σ_t is the predicted standard deviation (volatility) of returns at time t. It measures how much the returns can deviate from the mean return, indicating the level of risk or uncertainty.
  - z_α is the z-score corresponding to the desired confidence level α. For a 95% confidence level, z_α is approximately 1.65. This value comes from the standard normal distribution and represents how many standard deviations away from the mean cover 95% of possible outcomes.
  - $VaR_t(\alpha)$ represents the Value-at-Risk at time t for the confidence level α. It is the maximum expected loss not to be exceeded with a certain confidence level over a defined period.