Distributional Forecasting of Stock Price Change using Long Short-Term Memory and Mixture Density Network Layers

Ádám Szabados

May 2024

Abstract

This paper investigates the application of deep learning techniques, specifically Long Short-Term Memory (LSTM) networks augmented by the use Mixture Density Network (MDN) layers, for predicting stock price changes. With the increasing complexity and volatility of financial markets, accurate forecasting of stock price movements has become a crucial task for investors and financial analysts. Traditional methods often struggle to provide viable results on unseen data.

In this study, we propose a novel approach that combines the powerful memory capabilities of LSTM networks with the probabilistic modeling flexibility offered by MDN layers. By incorporating MDN layers into the architecture, our model is able to capture the nature of stock price distributions, allowing for more robust predictions and better handling of uncertainty.

We evaluate the performance of our proposed model using historical stock market data and simulate it with backtesting on data that the model hasn't seen before. Experimental results demonstrate that the LSTM-MDN model consistently provides a viable solution in terms of predictive accuracy and uncertainty estimation.

Overall, our findings suggest that LSTM networks with MDN layers hold promise as a powerful tool for predicting stock price changes, offering valuable insights for investors and financial practitioners seeking to make informed decisions in dynamic market environments.

Contents

1 Introduction

The prediction of stock price movements has long been an instrumental challenge in financial analysis and investment decision-making. Accurate forecasting of stock prices is essential for investors, traders, and financial institutions to optimize portfolio management, mitigate risk, and capitalize on market opportunities. Traditional models and statistical techniques have been widely employed for this purpose, but they often struggle to capture the complex and nonlinear dynamics in financial time series data.

In recent years, there has been a growing interest in applying machine learning (ML) and deep learning (DL) techniques to stock price prediction tasks. These methods offer the potential to capture intricate patterns and dependencies in data, providing more accurate and robust forecasts compared to traditional approaches. Among the various ML and DL models, Long Short-Term Memory (LSTM) networks have emerged as a popular choice for modeling sequential data due to their ability to effectively capture long-range dependencies contextually.

However, despite their success, LSTM networks have limitations, particularly in modeling the uncertainty inherent in stock price predictions. Stock prices are influenced by a multitude of factors, including economic indicators, geopolitical events and market dynamics, leading to uncertainty in their future movements. Addressing this uncertainty is crucial for making informed investment decisions and managing risk effectively.

To overcome these limitations, we propose a robust model that combines LSTM networks with a Mixture Density Network (MDN) layer. MDNs offer a probabilistic solution for modeling complex distributions, allowing for more accurate estimation of uncertainty in predictions. By integrating MDN layers into the LSTM architecture, our model aims to capture both the deterministic trends and the stochastic nature of stock price changes, providing more reliable forecasts and better risk assessment.

In this paper, we present a comprehensive study of the proposed LSTM-MDN model for predicting stock price changes. The rest of the paper is organized as follows: Section 2 lays down the basic theoretics behind neural networks. Section 3 presents the methodology, including the architecture of the LSTM-MDN model and the data preprocessing steps. Section 4 describes the application of our LSTM-MDN model to predict stock prices. Section 5 provides the results of the evaluated model. Finally, Section 6 concludes the paper with a summary of key findings and directions for future research.

2 Theoretical Basics

In this section, we provide an introductory overview of the theoretical foundations underlying neural networks, with a focus on Long Short-Term Memory (LSTM) networks and Mixture Density Networks (MDNs). Understanding these concepts is fundamental for grasping the basics of the proposed model for stock price prediction.

2.1 Artificial Neural Networks (ANN)

Artificial Neural Networks are a class of machine learning models inspired by the structure and functioning of the human brain. At their core, neural networks consist of interconnected nodes, or neurons , organized into layers. Each neuron receives input signals, performs a computation, and passes the result to the neurons in the next layer, making it a Feedforward Network. Through a process known as training, neural networks learn to adjust the strengths of connections between neurons (weights) to map input data to output predictions.

Let's denote the input to a neural network as x, where x is a vector representing the features of the input data. The output of a neural network is denoted as y, which could be a scalar value (in regression tasks) or a vector of probabilities (in classification tasks).

The computation performed by each neuron can be mathematically expressed as follows:

$$z = \sum_{i=1}^{n} w_i * x_i + b$$

Where x_i are the input values, w_i are the weights associated with each input and b is the bias term.

This weighted sum is then passed through an activation function $f(z)$ to introduce nonlinearity into the neuron's output.

2.2 Activation function

In this section, we are going to present a brief overview of some of the common activation functions and their properties. Activation functions introduce non-linearity into neural networks, enabling them to learn complex patterns and relationships within data. The choice of activation function depends on the specific task and architectural considerations of the neural network.

Sigmoid

The sigmoid activation function, also known as the logistic function, squashes the input into the range [0, 1]. It is commonly used in the output layer of binary classification tasks. Its mathematical formulation is the following:

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

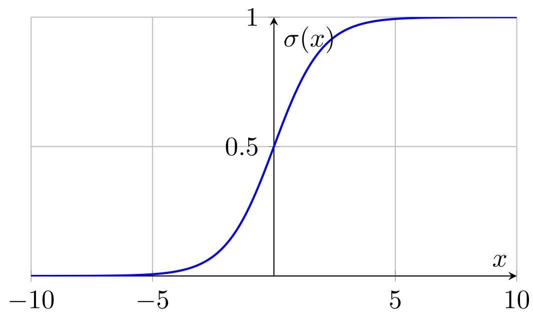

Figure 1: Plot of the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

Hyperbolic Tangent (Tanh)

The hyperbolic tangent function is similar to the sigmoid function but squashes the input into the range [-1, 1]. It is commonly used in hidden layers of neural networks. The mathematical representation is as follows:

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

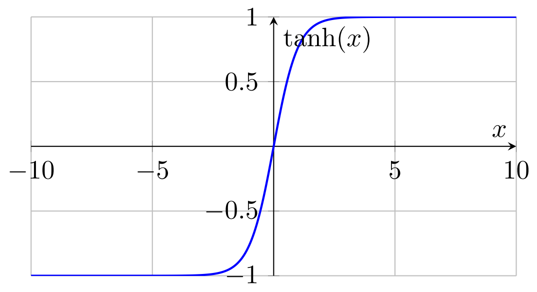

Figure 2: Plot of the hyperbolic tangent function $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Rectified Linear Unit (ReLU)

The ReLU activation function introduces non-linearity by setting negative inputs to zero and leaving positive inputs unchanged. It has become the default choice for many neural network architectures due to its simplicity and effectiveness. Mathematically, we can describe the ReLU function in the following way:

$$\text{ReLU}(z) = \max(0, z)$$

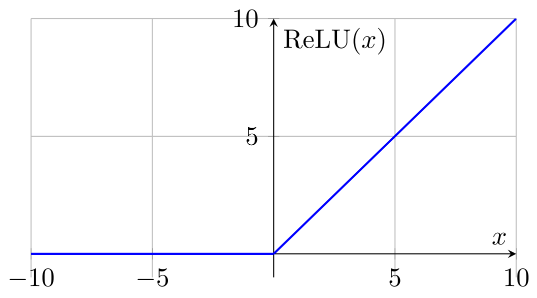

Figure 3: Plot of the ReLU function $\text{ReLU}(x) = \max(0, x)$

Leaky ReLU

The Leaky ReLU activation function is a variant of ReLU that allows a small, non-zero gradient for negative inputs. This helps address the "dying ReLU" problem where neurons with negative inputs always output zero and stop learning:

$$\text{LeakyReLU}(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha z, & \text{otherwise} \end{cases}$$

Where z is the input of the neuron, and α is a small constant (e.g., 0.01) controlling the slope of the function for negative inputs.

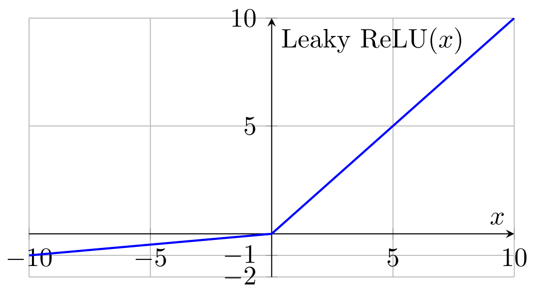

Figure 4: Plot of the Leaky ReLU function Leaky ReLU$(x) = \max(0.1x, x)$

Softmax

The softmax activation function is commonly used in the output layer of multi-class classification tasks. It converts raw scores into probabilities, ensuring that the sum of the probabilities across all classes is equal to 1:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where z_i is the raw score for class i, and K is the total number of classes.

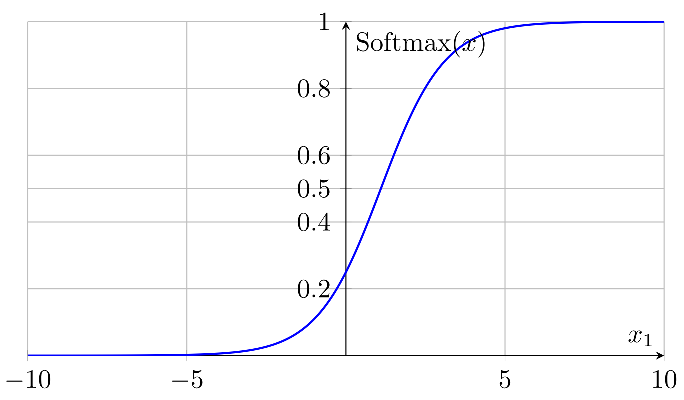

Figure 5: Plot of the Softmax function

2.3 Network architecture

Neural networks are typically organized into layers: an input layer, one or more hidden layers, and an output layer. The input layer receives the raw input data, the hidden layers perform computations, and the output layer produces the final predictions.

Each neuron in a layer is connected to every neuron in the subsequent layer, forming a fully connected, or dense, architecture. This allows neural networks to capture complex relationships between features in the data.

2.4 Training the network

During the training phase, neural networks learn to adjust their weights and biases to minimize a predefined loss function, which measures the difference between the predicted output and the true output. This process is typically done using optimization algorithms such as stochastic gradient descent (SGD) or variants like Adam.

The weights and biases are updated iteratively using the gradients of the loss function with respect to the network parameters, obtained by backpropagation. This process continues until the model converges to a set of parameters that provide satisfactory performance on the training data.

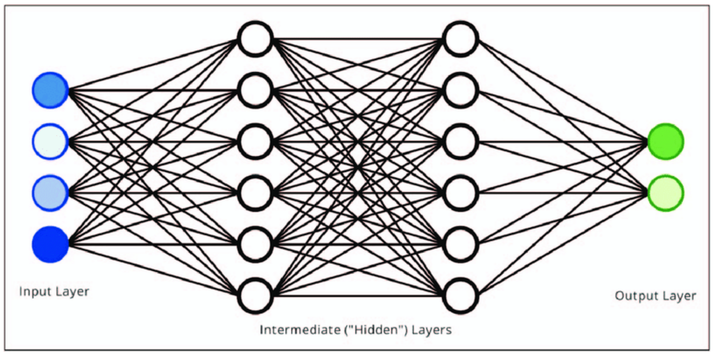

Figure 6: Structure of a Feedforward Neural Network [1]

2.5 Loss function

Loss functions, also referred to as cost functions, serve to measure the difference between the predicted outputs of a neural network and the actual labels. They play an important role in assessing the model's performance on the training data and guiding the optimization process during training.

There is a wide variety of loss functions for deep learning. In this section, we will introduce two of the most commonly used functions, and lastly the function that will be used for training our Mixture Density Network (MDN).

Mean Squared Error (MSE)

The Mean Squared Error (MSE) stands out as one of the most prevalent loss functions, particularly in regression tasks. It calculates the average squared difference between the predicted and actual values.

Mathematically, for each sample i, with a prediction $\hat{y}_i$ and a corresponding true label y_i, the MSE loss is computed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Where n denotes the total number of samples.

Binary Cross-Entropy

The Binary Cross-Entropy Loss, commonly known as Log Loss, finds its application in binary classification tasks. It evaluates the similarity between two probability distributions - the predicted probabilities and the true binary labels.

For a binary classification problem, with predictions $\hat{y}_i$ and true labels y_i, the Binary Cross-Entropy Loss is mathematically represented as:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Where N denotes the total number of samples.

Negative Log Likelihood

Negative Log Likelihood (NLL) is another prominent loss function, especially in probabilistic models and tasks such as maximum likelihood estimation. It measures the negative log-likelihood of the observed data given the model parameters.

Minimizing Negative Log Likelihood is equivalent to maximizing the likelihood of observing the training data under the model. In other words, it drives the model parameters towards values that make the observed data most probable according to the model.

For a probabilistic model with predicted probabilities $\hat{y}_{ij}$ for class j and sample i, and corresponding true probability distributions y_{ij}, the Negative Log Likelihood can be computed in by using the following formula:

$$NLL = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

2.6 Backpropagation

Backpropagation is a key algorithm for training neural networks, leveraging the chain rule to compute gradients of the loss function with respect to the model parameters. Through this process, the network learns to adjust its weights and biases to minimize the loss and improve its predictive performance. Backpropagation is made up of forward pass and backward pass.

Forward pass

During the forward pass, the neural network computes predictions by propagating the input data through its layers. Mathematically, for a given input x, the output of the network can be expressed as:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$
$$\mathbf{a}^{(l)} = f^{(l)}(\mathbf{z}^{(l)})$$

Where $\mathbf{z}^{(l)}$ is the pre-activation output of layer l, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases of layer l, $\mathbf{a}^{(l)}$ is the activation output of layer l, $f^{(l)}$ is the activation function of layer l.

Backward pass

In the backward pass, gradients of the loss function $\mathcal{L}$ with respect to the model parameters are computed using the chain rule. The gradients are computed recursively from the output layer to the input layer using the following equations:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(L)}} = \nabla_{\mathbf{a}^{(L)}}\mathcal{L} \odot f'^{(L)}(\mathbf{z}^{(L)})$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}}(\mathbf{a}^{(l-1)})^T$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}}$$

Where L is the index of the output layer, $\nabla_{\mathbf{a}^{(L)}}\mathcal{L}$ is the gradient of the loss function with respect to the activation output of the output layer and $f'^{(L)}$ is the derivative of the activation function of the output layer.

In-depth insights into backpropagation can be explored further in reference [4].

2.7 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining hidden states that capture information about previous elements in the sequence. They are particularly suited for tasks such as time series prediction (in our case, stock price prediction), natural language processing, and speech recognition, where the input data has a temporal or sequential structure that serves as a context for the model.

In a standard RNN, each neuron in the network has a recurrent connection to itself, allowing it to maintain a state vector that captures information about the sequence that it has seen so far. Mathematically, the hidden state h_t at time step t is computed as a function of the current input x_t and the previous hidden state h_{t-1}:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_n)$$

Where f is the activation function (commonly ReLU or tanh), W_{hh} and W_{xh} are the weight matrices for the recurrent and input connections respectively and b_h is the bias vector.

While RNNs are effective at modeling sequential data, they suffer from the vanishing gradient problem, where gradients diminish exponentially over long sequences, leading to difficulties in learning long-range dependencies.

2.8 Long Short-Term Memory Networks (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized type of RNN designed to address the vanishing gradient problem and capture long-range dependencies more effectively. They achieve this by introducing additional gating mechanisms that control the flow of information through the network.

An LSTM unit consists of a cell state c_t and three multiplicative gates: the input gate, forget gate, and output gate. These gates regulate the flow of information into and out of the cell state, allowing the LSTM to selectively remember or forget information over time. The input gate controls how much of the new information from the current input and the previous hidden state should be added to the cell state. It decides which values will be updated by regulating the extent to which new information will influence the cell state. The forget gate determines which information from the previous cell state should be discarded. It plays a crucial role in resetting or maintaining parts of the cell state that are no longer relevant to future predictions. The output gate controls how much of the cell state should be exposed to the next hidden state. It regulates the extent to which the cell state contributes to the output of the LSTM unit.

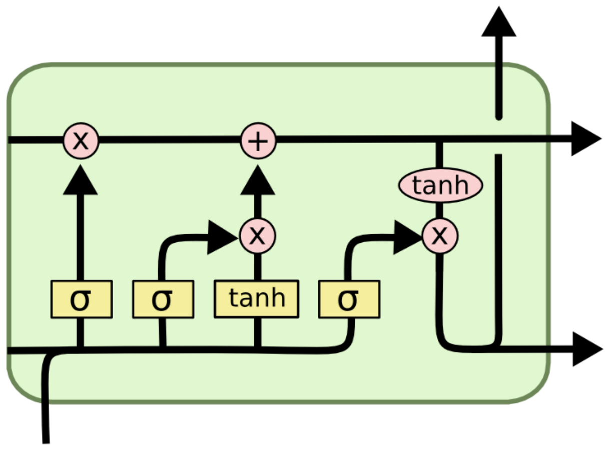

Figure 7: Structure of an LSTM network [2]

The computations within an LSTM unit can be described by the following equations:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
g_t &= \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Where i_t, f_t and o_t are the input, forget and output gates respectively, g_t is the cell state, σ is the sigmoid activation function and $\odot$ represents element-wise multiplication.

LSTM networks are powerful tools for modeling sequential data and capturing temporal dependencies. Their ability to effectively process time series data has led to significant advancements in various fields, and the stock market is no different. LSTM networks will allow our model to learn stock market data in a context. Giving an LSTM network a shorter context will allow it to pick up on shorter trend changes, which could be useful when trading more volatile stocks, like Forex Stocks. Similarly, using a longer context will allow our model to learn trend changes on a broader scale.

2.9 Mixture Density Network (MDN)

Mixture Density Networks (MDNs) are a powerful extension of neural networks designed to model complex, multimodal probability distributions. Unlike traditional neural networks that typically output estimates, MDNs output parameters of a mixture model, allowing them to capture uncertainty and multiple modes in the data. This capability is particularly useful in tasks where the target variable can take on multiple possible values for a given input.

An MDN consists of a standard neural network followed by an output layer that parameterizes a mixture of probability distributions. Typically (and in our case), Gaussian mixtures are used, but other types of distributions can be used depending on the application. The network learns to output the parameters of these distributions, including the means, variances, and mixing coefficients. The shape of data we get from the output layer depends on the number of distributions we want to output to form the predicted mixture distribution.

For a given input x, the MDN outputs the parameters of a mixture of M Gaussian components. These parameters are the mixing coefficients π_i, the means μ_i and the variances σ_i^2

Mathematically, the output of the MDN can be expressed in the following way:

$$p(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{M} \pi_i(\mathbf{x}) \mathcal{N}(\mathbf{y}|\mu_i(\mathbf{x}), \sigma_i^2(\mathbf{x}))$$

Where $\mathcal{N}(\mathbf{y}|\mu_i), \sigma_i^2)$ is the probability density function of the i-th Gaussian component, $\pi_i(\mathbf{x})$ are the mixing coefficients, which sum to 1 and are obtained using the softmax function, $\mu_i(\mathbf{x})$ are the means of the Gaussian components, $\sigma_i^2(\mathbf{x})$ are the variances of the Gaussian components, constrained to be positive using an exponential.

After the output is computed, the output parameters are computed the following way:

$$\pi_i(\mathbf{x}) = \frac{\exp(z_{\pi_i})}{\sum_{j=1}^{M} \exp(z_{\pi_j})}$$

$$\mu_i(\mathbf{x}) = z_{\mu_i}$$

$$\sigma_i(\mathbf{x}) = \exp(z_{\sigma_i})$$

Where z_{π_i}, z_{μ_i} and z_{σ_i} are the raw outputs from the neural network.

Training an MDN involves maximizing the likelihood of the observed data under the predicted mixture distribution. As menioned before, this is equivalent to minimizing the negative log-likelihood (NLL) of the data. In our case, the model will be predicting the mixture density function for the change of the Close Price for a given stock (Delta Close Price)

3 Preparing the model

The preparation of the model involves several steps, including data acquisition, preprocessing, feature selection, data scaling, and splitting the data into training and testing sets. For our study, we utilize stock price data provided by Yahoo Finance, which offers historical stock prices and associated financial metrics.

Data Description

The data provided by Yahoo Finance includes various attributes that are essential for analyzing stock price movements and conducting predictive modeling. These attributes are obtained for each trading day and include:

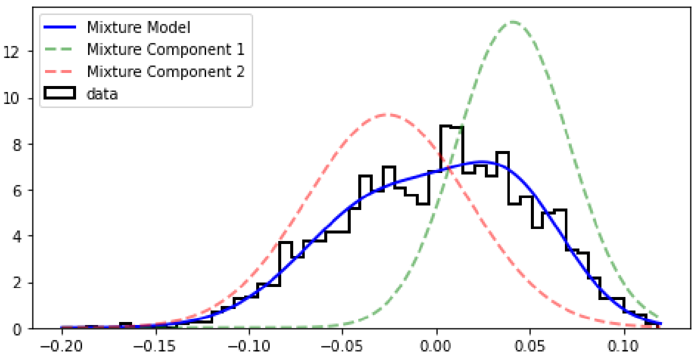

Figure 8: Example of a Mixture Density Network [3]

- Open Price: The price of a stock at the beginning of the trading day.
- High Price: The highest price of the stock during the trading day.
- Low Price: The lowest price of the stock during the trading day.
- Close Price: The price of the stock at the end of the trading day.
- Adjusted Close Price: The closing price of the stock adjusted for factors such as dividends, stock splits, and corporate actions, providing a more accurate reflection of the stock's value.
- Volume: The total number of shares traded during the trading day.

While the data obtained from Yahoo Finance provides a solid foundation for our analysis, we aim to enhance our predictive model by incorporating additional data fields and technical indicators. These features offer deeper insights into stock price movements and provide more accurate predictions.

In addition to the standard fields of open, high, low, close, adjusted close prices, and volume, we will leverage the following data and indicators:

- Log-Return: Logarithmic returns are a commonly used metric in finance for measuring the percentage change in a stock's price over a given time period. They provide a more robust measure of return compared to simple percentage changes and are particularly useful for analyzing price dynamics over multiple time periods.
- Relative Strength Index (RSI): RSI is a momentum oscillator that measures the speed and change of price movements. It oscillates between 0 and 100 and is typically used to identify overbought or oversold conditions in a stock. RSI values above 70 typically indicate overbought conditions, while values below 30 suggest oversold conditions.
- Simple Moving Average (SMA): SMA is a widely used technical indicator that calculates the average price of a security over a specified number of periods. It smooths out price data to identify trends and is useful for determining support and resistance levels.
- Exponential Moving Average (EMA): EMA is a type of moving average that gives more weight to recent prices, making it more responsive to recent price changes compared to SMA. It is particularly useful for capturing short-term trends and momentum.
- Weighted Moving Average (WMA): WMA assigns weights to each price data point, with more recent prices receiving higher weights. Like EMA, WMA is sensitive to recent price changes and is useful for identifying short-term trends.
- Bollinger Bands: Bollinger Bands are a volatility indicator consisting of three lines plotted above and below a central moving average. The bands widen during periods of high volatility and narrow during periods of low volatility, providing insights into potential breakout or reversal points in the price trend.

Features with no range

In predictive modeling, it is often beneficial to store the changes of certain features rather than the absolute values, especially for features that do not have a fixed minimum and maximum value. This practice helps to normalize the data and mitigate the influence of outliers or extreme values, making the model more robust and interpretable.

For moving averages such as Simple Moving Average (SMA), Exponential Moving Average (EMA), and Weighted Moving Average (WMA), which do not have predefined upper or lower bounds, storing the changes (DeltaSMA, DeltaEMA, DeltaWMA) instead of the absolute values can provide several advantages, while features like the Relative Strength Index (RSI) or the Bollinger Bands Percentage (BBP) can be stored as they are.

Scaling the data

When it comes to training neural networks, scaling the data is a crucial preprocessing step that helps ensure uniformity and stability in the training process. Scaling involves transforming the features so that they have a consistent scale, typically with a mean of zero and a standard deviation of one. This normalization process is essential for algorithms that rely on distance metrics or gradient descent optimization, as it prevents features with larger scales from dominating the training process. For our model, we will be using standard scaling, also known as Z-score normalization, which is one of the most commonly used scaling techniques. It transforms the data such that it has a mean of zero and a standard deviation of one. Mathematically, each feature x_i is scaled according to the following formula:

$$z_i = \frac{x_i - \mu}{\sigma}$$

Where z_i is the scaled value of feature x_i, μ is the mean of the feature x_i and σ is the standard deviation of feature x_i.

3.1 Describing the model

In this section, we provide an overview of the architecture and configuration of our neural network to make the predictions.

Our LSTM network is configured to use a 21-unit wide window, meaning it takes 21 days of stock data as input for each prediction. This window size allows the model to capture significant short-term temporal patterns and dependencies in the stock price movements. The final layer of the network is a mixture density network (MDN) layer, which outputs the parameters of two Gaussian distributions. These parameters include the means, standard deviations, and mixing coefficients of the Gaussian components.

We use Negative Log Likelihood (NLL) as the loss function for our model. NLL is suitable for training mixture density networks as it measures the likelihood of the observed data under the predicted probability distributions. The objective is to maximize the likelihood, which is equivalent to minimizing the negative log likelihood.

To prevent overfitting and ensure generalization, we apply L1 regularization to the model. L1 regularization adds a penalty proportional to the absolute value of the coefficients, encouraging sparsity in the model weights.

For optmization, we use the Adam optimizer with a learning rate of 0.001. Adam is an adaptive learning rate optimization algorithm that combines the advantages of two other extensions of stochastic gradient descent: adaptive gradient algorithm (AdaGrad) and root mean square propagation (RMSProp). It is well-suited for handling sparse gradients on noisy problems.

To further prevent overfitting and optimize training efficiency, we employ early stopping. This technique monitors the validation loss during training and stops the training process if the validation loss does not improve for a specified number of epochs, thereby preserving the best model state.

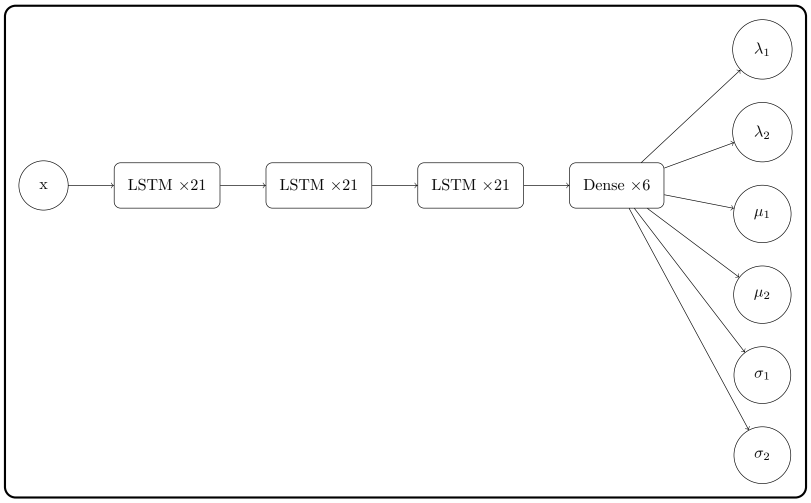

Figure 9: Architecture of the neural network model

4 Applications

Once the model is trained, it can be applied to predict future stock price movements. Specifically, our LSTM-MDN model outputs the parameters of a mixture distribution, which provides a probabilistic forecast of the delta close price (the change in the closing price).

The MDN layer in our model produces a mixture of two Gaussian distributions, characterized by their means, standard deviations, and mixing coefficients. To make trading decisions based on these predictions, we calculate the probability that the delta close price will increase.

Mathematically, we can integrate the predicted mixture distribution from 0 to infinity to obtain the probability that the delta close price ($\Delta Close$) will be greater than 0:

$$P(\Delta\text{Close} > 0) = \int_0^\infty p(\Delta\text{Close})\, d(\Delta\text{Close})$$

Trading algorithm

Based on the calculated probability $P(\Delta\text{Close} > 0)$, we apply the following trading algorithm:

1. **Buy Signal:** If $P(\Delta\text{Close} > 0) > 0.5$, we interpret this as the odds favoring an increase in the delta close price, and we generate a buy signal.
2. **Sell Signal:** If $P(\Delta\text{Close} > 0) < 0.5$, we interpret this as the odds favoring a decrease in the delta close price, and we generate a sell signal.
3. **Hold Signal:** If $P(\Delta\text{Close} > 0) \approx 0.5$, indicating uncertainty or an equal likelihood of increase and decrease, we generate a hold signal.

The decision-making process can be summarized as follows:

$$\text{Decision} = \begin{cases} \text{Buy} & \text{if } P(\Delta\text{Close} > 0) > 0.5 \\ \text{Sell} & \text{if } P(\Delta\text{Close} > 0) < 0.5 \\ \text{Hold} & \text{if } P(\Delta\text{Close} > 0) \approx 0.5 \end{cases}$$

5 Results

To evaluate the performance of our predictive model, we implemented a backtesting framework. For training, we used all available historical data up until exactly one year before our current date. The backtesting period began at the start of that date, providing each stock a full year to trade using the algorithm described earlier.

The table below summarizes the performance of our model on four selected stocks: NVDA, TSLA, AAPL, and GOOG. The metrics include the training loss, validation loss, and the return over the backtesting period.

Stock	**loss**	**val loss**	**Return**
NVDA	0.18	0.25	+39%
TSLA	-0.29	0.30	+90%
AAPL	0.52	0.78	+120%
GOOG	0.28	0.89	+43%

Table 1: Performance metrics of backtesting

The results indicate that our LSTM-MDN model, combined with the trading algorithm, was able to generate positive returns for most of the stocks during the backtesting period. The validation loss provides insight into the model's ability to generalize to unseen data, while the returns demonstrate the practical effectiveness of the model in a trading scenario.

6 Conclusion and Future Improvements

The results from our backtesting procedure demonstrate that the model is viable and has the potential to generate positive returns.

The LSTM-MDN model proved capable of capturing temporal dependencies in stock price data and providing probabilistic predictions for future price movements. Our trading algorithm, which utilized these predictions, was able to generate positive returns for the stocks mentioned above.

Future work could explore incorporating additional features and indicators that may capture more complex market dynamics. This could include more sophisticated technical indicators or sentiment analysis from news or social media.

The evaluation of the model could also be improved by training and backtesting the model on a year-to-year basis. Instead of using all available historical data for training and then testing on a single year, the model could also be trained on data up to the start of each year and then tested on the next year. This rolling window approach would better mimic real-world trading scenarios and provide a more thorough evaluation of the model's performance over time.

Further tuning of hyperparameters, such as the learning rate, the number of LSTM units, and the complexity of the MDN layer, could lead to improved performance. Additionally, experimenting with different neural network architectures, such as Attention mechanisms or Transformer models, could provide better representations of temporal dependencies.

References

[1] Basic design of a neural network. Adapted from 'Network.svg' by Victor C. Zhou (https://victorzhou.com/series/neural-networks-from-scratch)

[2] Christopher Olah. Understanding LSTM Networks, 2015.

[3] Kaneel Senevirathne. Forecasting Stock Market Trends: A Mixture Model Approach with TensorFlow Probability, 2023

[4] Si Cheng Fong, Distributional Prediction of Foreign Exchange Rates with Mixture Density Network. Master's thesis, Imperial College London. 2021.