509 Project

Stephanie Daniella Hernandez Prado
Nanzhu Li
Serena Zhou

April 1, 2025

1. Introduction

1.1 Background and Motivation

Traditional financial theories, such as the Efficient Market Hypothesis (EMH), assert that asset prices fully reflect all available information, making it impossible to consistently predict future movements based on past data. However, behavioral finance challenges this view by identifying market anomalies and cognitive biases that influence investor behavior. One of the most widely observed phenomena is the **momentum effect**, where assets that have performed well in the recent past tend to continue performing well in the short to medium term, and vice versa for underperforming assets. This effect contradicts the EMH and suggests that investors may underreact to new information or follow patterns without reassessing fundamental values.

This behavioral perspective is supported by psychological biases such as **confirmation bias**, where investors selectively focus on information that confirms their existing beliefs, and **herding behavior**, where individuals follow market trends driven by group actions rather than independent analysis. Together, these biases can reinforce price trends and create opportunities for predictive modeling based on historical return data.

Given these insights, this project explores a neural network approach for multi-class trend prediction using technical indicators derived from past returns. The goal is to capture potential momentum patterns in financial time series while acknowledging the behavioral forces that may drive them.

3. Data

3.1 Data Collection

We collected historical stock data for **Apple Inc. (AAPL)** using the `yfinance`. The dataset spans from **January 1, 2012 to March 15, 2025**, with data sampled at a daily interval. The raw dataset includes the following features:

- Open, High, Low, Close prices
- Adjusted Close price

- Daily trading volume

This financial time series serves as the basis for return calculations and trend prediction.

Data Processing

a) Return Calculation
Daily returns were computed using the percentage change in adjusted closing prices:

$$\text{Return}_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \texttt{pct\_change()}$$

b) Features
To capture short-term return dynamics, we created sliding windows of **15 consecutive daily returns**. Each window forms a single input sample to the neural network:

$$\begin{aligned} \text{Sample}_1 &= [r_1, r_2, \ldots, r_{15}] \\ \text{Sample}_2 &= [r_2, r_3, \ldots, r_{16}] \\ &\vdots \\ \text{Sample}_{3304} &= [r_{3302}, r_{3303}, \ldots, r_{3304}] \end{aligned}$$

This resulted in a dataset of **3,304 samples**, each consisting of **15 features**.

c) Label
For each sample, we calculated the sum of returns over the following 5 days and labeled the trend based on a threshold defined as $0.3 \times$ standard deviation of future returns. The target variable was then classified into three categories:

- **0**: Downtrend (future return $< -$threshold)
- **1**: Neutral (within $\pm$threshold)
- **2**: Uptrend (future return $>$ threshold)

Thus, each row in the final dataset consists of a 15-day return window (input) and a 3-class trend label (output).

4. Model Design and Implementation

4.1 Neural Network (Mathematical)

A neural network (ANNs) is a computational model composed of layers of interconnected nodes that are capable of learning nonlinear mappings between input features and target outputs. Inspired by the structure of biological neurons, each artificial neuron performs a weighted sum of its inputs followed by a nonlinear activation function. Through a process of iterative optimization using labeled data, neural networks are able to learn complex patterns and decision boundaries that traditional linear models may fail to capture.

The most basic type of neural network is the feedforward neural network, also known as the multilayer perceptron (MLP). In this architecture, data flows in one direction—from the input layer

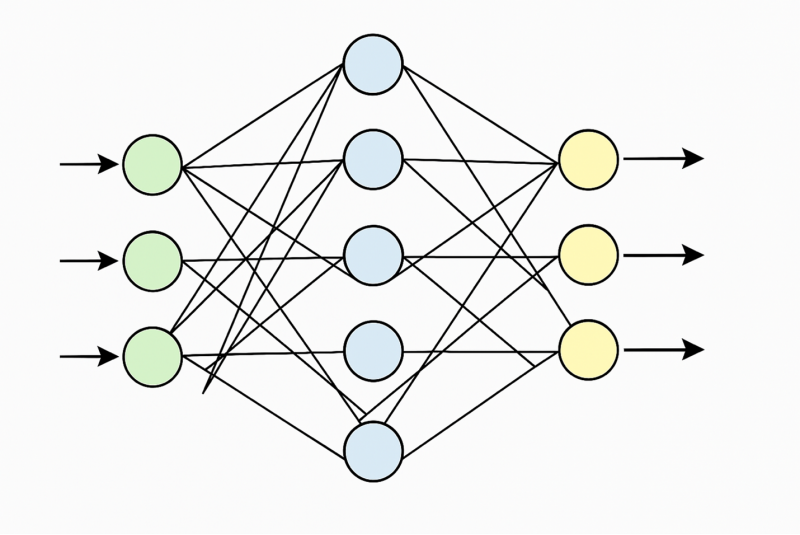

Figure 1: Feedforward Neural Network Architecture

through one or more hidden layers to the output layer—without any recurrent or convolutional connections. Each neuron in a given layer is fully connected to all neurons in the subsequent layer.

In classification tasks, the final layer of the network typically uses a softmax activation function to convert the outputs into a probability distribution over predefined classes. During training, the model parameters (weights and biases) are optimized to minimize a loss function such as categorical or sparse categorical cross-entropy using a gradient-based algorithm, such as the Adam optimizer.

Modern feedforward networks often include additional components to enhance training dynamics and generalization capability:

- **Batch Normalization:** Stabilizes learning by normalizing inputs to each layer.
- **Dropout:** Reduces overfitting by randomly deactivating neurons during training.
- **LeakyReLU Activation:** Addresses the vanishing gradient problem by allowing small gradients when inputs are negative.

Neural networks are particularly suitable for modeling financial data such as return sequences, where complex, nonlinear temporal relationships may exist. Although convolutional or recurrent architectures are often used for sequence modeling, fully connected networks remain effective and interpretable for structured time-series data.

4.2 Labeling Strategy

Follow something similar to the normal distribution: Threshold = $\pm 0.3 \times$ standard deviation Classification into **Down / Neutral / Up**

Before modeling, we first transformed the raw stock price data into a more informative representation. We calculated the daily log returns from the adjusted closing prices of AAPL stock, covering the period from January 1, 2012, to March 15, 2025. The log return for day t is given by:

$$R_t = \log\left(\frac{P_t}{P_{t-1}}\right)$$

where P_t is the adjusted closing price at time t. This transformation reduces the impact of absolute price scale and focuses on relative percentage change, which is more suitable for modeling financial time series.

To evaluate the distributional properties of the return series, we plotted a **Q–Q plot** comparing the empirical quantiles of daily returns to those of a standard normal distribution.

Despite the departure from perfect normality, we proceeded under the assumption that the **rolling sum of returns over a short horizon (e.g., 5 days)** is approximately normally distributed. This is justified by the **Central Limit Theorem**, which states that the sum of weakly dependent variables tends toward a normal distribution as the sample size increases:

$$S_t = \sum_{i=1}^{5} R_{t+i}$$

To construct trend labels for supervised learning, we defined the following rolling return sum threshold:

$$\theta = 0.3 \times \sigma_S$$

Then, each sample was categorized into one of three classes based on the following rule:

- **Uptrend (2):** if $S_t > \theta$
- **Downtrend (0):** if $S_t < -\theta$
- **Neutral (1):** if $-\theta \leq S_t \leq \theta$

This classification rule creates a buffer zone around zero and ensures that only statistically significant upward or downward movements are labeled as trends, while small fluctuations are treated as neutral.

To evaluate this labeling method, we applied the resulting labels to the original price data and visualized them by color-coding each region according to the assigned class. The visualization confirms that the labels correspond well to actual market movements.

4.3 Model Implementation and Training

Based on the labeled dataset constructed in Section 4.2, we implemented a fully connected feed-forward neural network to predict the trend class of each sample. The model was built using the Keras `Sequential` API with multiple hidden layers and nonlinear activations. The architecture was designed to balance expressiveness and generalization while remaining computationally efficient.

This section outlines the full implementation process of the classification model, including the neural network architecture, training procedure, loss function, optimizer configuration, and the final performance evaluation through both numerical and visual methods.

Figure 2: Historic AAPL Prices with Color-coded Trend Labels

4.3.1 Network Architecture

The model takes as input a 15-dimensional vector, which represents the daily returns from the past 15 trading days. This input is passed sequentially through three hidden layers before producing a probability distribution over the three possible trend classes.

The transformed outputs then enter the second hidden layer, which contains 20 neurons. Like the previous layer, each neuron is fully connected to all outputs from the prior layer. After applying another LeakyReLU activation, a dropout layer with a dropout rate of 0.25 is introduced to randomly deactivate a subset of neurons during training. This regularization technique reduces the risk of overfitting by encouraging the network to learn redundant representations.

The third hidden layer consists of 8 neurons, again fully connected to the 20 outputs from the preceding layer. As before, batch normalization is applied to the dense layer's output, followed by a LeakyReLU activation to maintain stable and nonlinear transformation. By this stage, the network has successfully transformed the original return sequence into a more abstract representation, progressively filtering and combining features relevant to trend prediction.

Finally, the output layer contains 3 neurons, corresponding to the three target classes: downtrend (0), neutral (1), and uptrend (2). A softmax activation function is applied to convert the raw outputs into class probabilities. The predicted class is then selected as the one with the highest probability. This structure allows the network to learn nonlinear decision boundaries in the return space and effectively distinguish between different market conditions.

4.3.2 Training Configuration

The dataset contained 3304 samples in total. It was split into training and test sets using an 80:20 ratio. Input features were standardized using `StandardScaler` to zero mean and unit variance, which is important for ensuring efficient training with batch normalization and gradient-based optimizers.

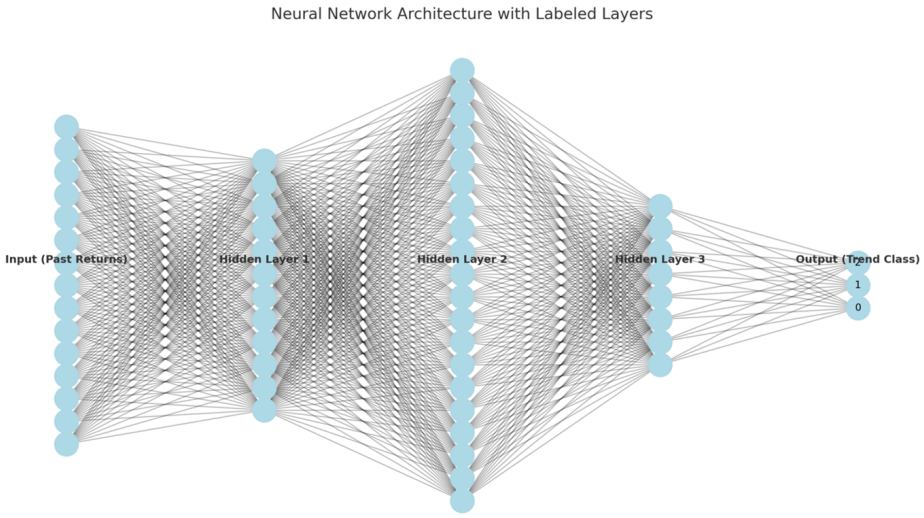

Figure 3: Neural Network Architecture with Labeled Layers

The model was trained using the following configuration:

- **Loss Function:** Sparse categorical cross-entropy
- **Optimizer:** Adam
- **Learning Rate:** 0.001
- **Batch Size:** 16
- **Epochs:** 50
- **Random Seed:** 996

Training was performed using mini-batch gradient descent with the Adam optimizer.

Loss Function

To train the model for multi-class classification, we used the sparse categorical cross-entropy loss, which is appropriate when labels are provided as integer class indices rather than one-hot encoded vectors. The loss function is defined as:

$$L = -\sum_{i=1}^{N} \log(p_{i,y_i})$$

where N is the number of samples in a batch, $y_i \in \{0, 1, 2\}$ is the true class label for the i-th sample, and p_{i,y_i} is the predicted softmax probability for the correct class.

This loss penalizes the model when it assigns low probability to the correct class, encouraging it to become more confident in correct predictions.

Appendix

Source Code Repository

All implementation code and related materials can be found at our GitHub repository:

`https://github.com/Stephanie-Daniella/Stock-Returns-Neural-Nerwork`