AI Stock Price Prediction
Daisy Zheng

Nowadays, financial investment analysts help companies and stakeholders predict future stock prices based on historical trends and other factors. However, we can let neural networks in AI do this job as an attempt to reduce human efforts and make accurate predictions more efficiently.

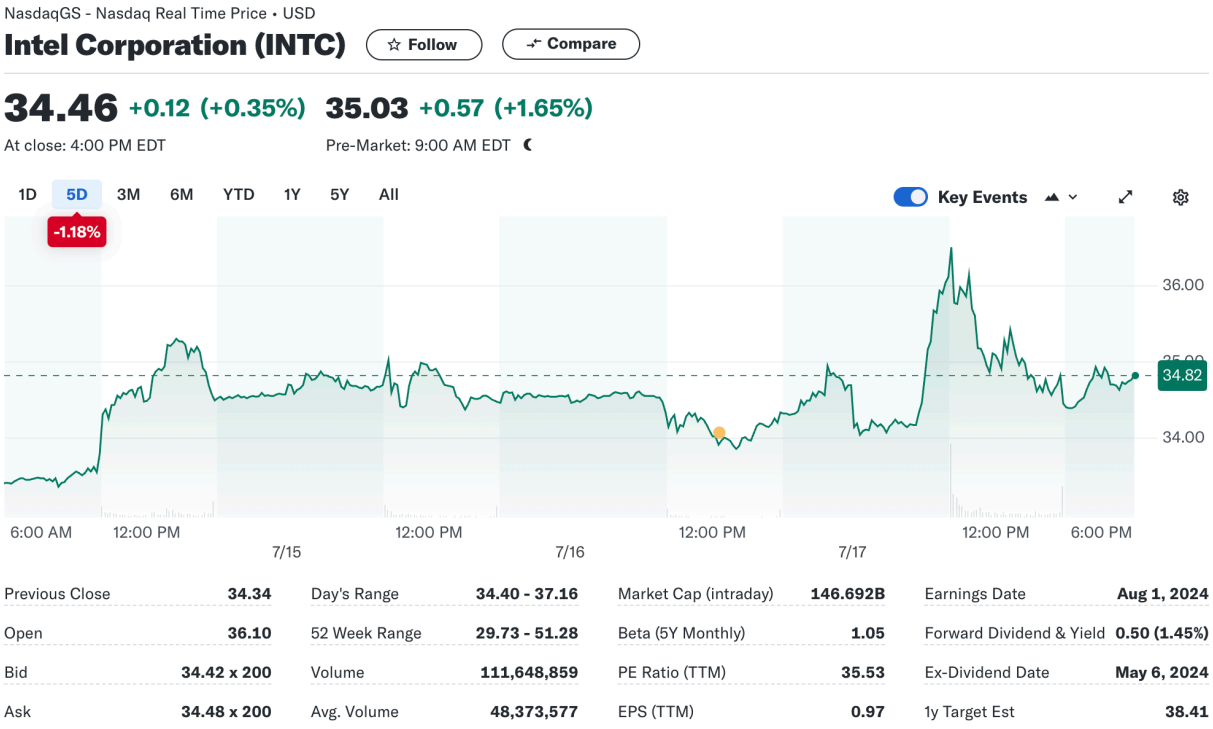


Part 1: Data Acquisition

Many existing neural networks learn from historical data and predict new ones, with the RNN and traditional neural networks being outstanding models for stock price predictions. But we first need to understand and acquire many stock metrics, which constantly impact stock prices, for our neural network to learn for the input. I chose 11 essential metrics as the input data for my neural network algorithm:

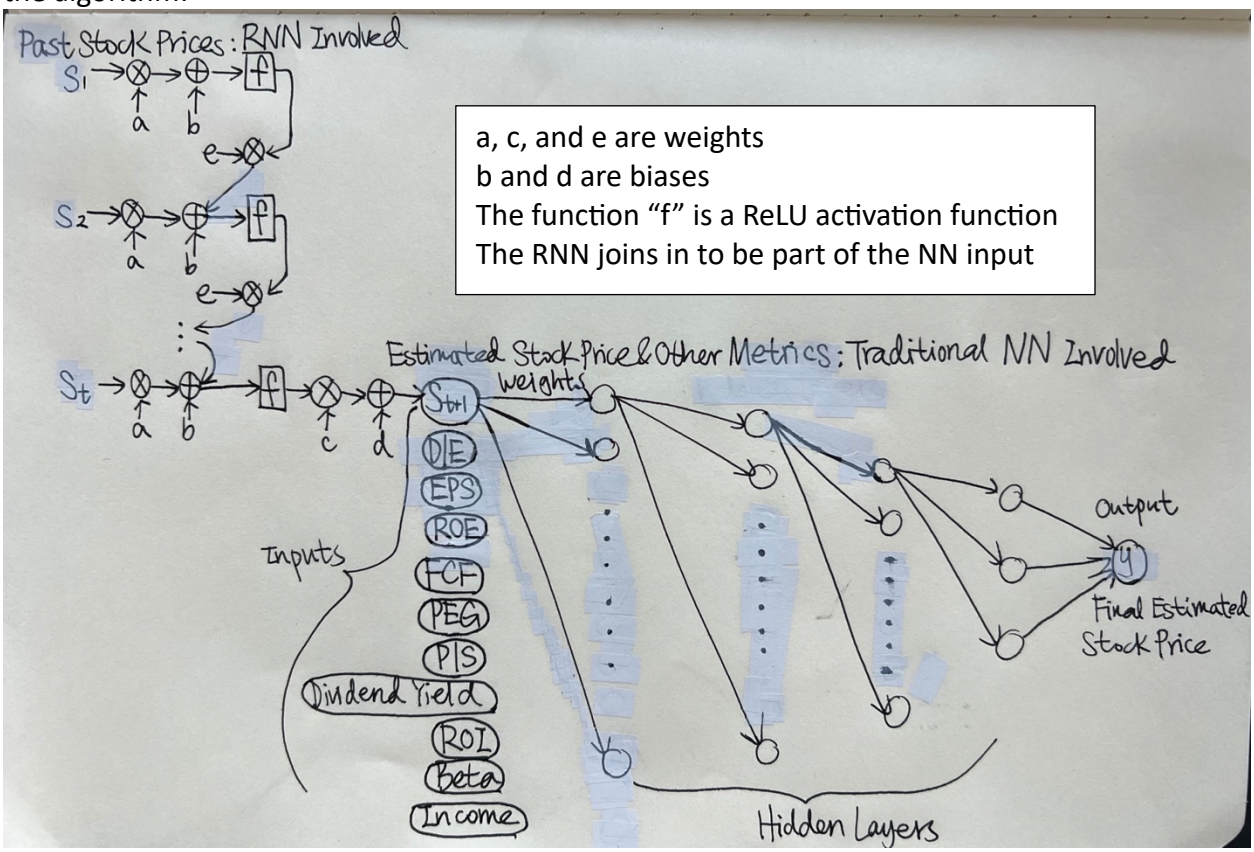
- Past stock prices: These are the prices in USD that a share of stock was trading for on the market in the past. They are the only input involving time-sequential data because stock prices were collected from the past several days.
- Debt-to-equity: This ratio shows how much of a company is owned by creditors compared with how much shareholder equity is held by the company. It impacts stock prices because investors use this metric to decide whether to invest in a company's stock.
- Earnings per share: This measures a company's profitability by dividing quarterly or annual income by the number of outstanding stock shares. It impacts stocks because the higher a company's EPS, the greater the profit and value investors perceive.

- Return on equity: It measures a company's net income divided by its shareholders' equity. It impacts stocks because the higher the ROE, the better a company is at converting its equity financing into profits; thus, more investors are interested in buying their stocks.
- Free cash flow: The money remains after a company pays for everyday operating expenses and capital expenditures. If it is positive, the company has excess cash to pay dividends, buy back stock, and pay down debt.
- PEG ratio: It measures the relationship between the price/earnings ratio and earnings growth. The PEG ratio builds off the price-to-earnings ratio and provides a deeper picture of stock value. PEG ratio impacts stocks because a PEG ratio of under 1.0 can indicate a stock is undervalued and a potential buy, while a PEG above 1.0 can indicate an overvalued stock.
- Price to sales: It is calculated by dividing a company's stock price by its sales per share – or revenue per share. It impacts stocks by comparing a company's stock price to its revenues, helping investors find undervalued stocks that make suitable investments.
- Dividend yield: It is the income derived from investing in a company's stock.
- Return on investment: It refers to the value that should be returned by investing in a company's stocks.
- Beta: It refers to an internal software testing activity that is open to the public. It is an important metric because it helps potential investors analyze the expected return on equity on the total amount pooled into such securities.
- Income: It is how much you get from the investment. It affects stock prices because strong earnings in income generally increase the stock price.



Part 2: Algorithm Pipeline

Now we have chosen the 11 significant stock metrics as inputs; we must move on to develop a functioning and effective algorithm for our neural network. Since there is time-sequential data in past stock prices and normal numerical data in other metrics, we have to combine and use two different neural networks to process input data and make accurate predictions. We will construct an RNN neural network for processing time-sequential data because it can compute complex patterns in data with time sequences. An RNN remembers past information, understands the temporal aspect of data, and learns from past experiences through function calculations while unrolling the inputs. After inputting past stock prices, we receive an estimated value of today's stock prices through the RNN as the output. We will then use this estimated stock price as the input for the stock price metric and combine it with the other input metrics in a traditional neural network, creating an accurate final output of the predicted stock price based on all the financial factors we go through. Here is a diagram of the entire process of the algorithm:



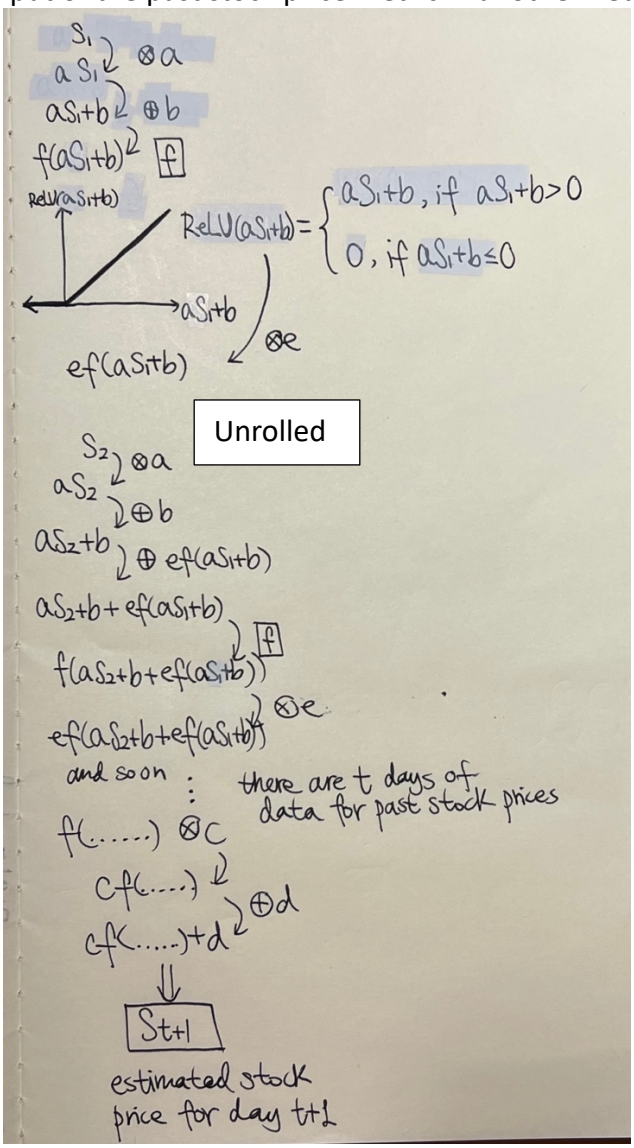
Step 1: RNN for past stock prices

The inputs S_1, S_2, …, S_t are the stock prices from the past few days of a company (S_1 is the oldest, S_t is the newest), and now we input them into an unrolled RNN model. In the first line, we first receive $f(aS_1 + b)$. "a" is a weight that adjusts the strength of the input to influence the signals transmitted across the network and ultimately determine the network's output. "b" is a bias that offsets the value of "aS_1" to shift the f towards the positive or negative side.

Then, $aS_1 + b$ is inputted into a ReLU activation function: f. If $aS_1 + b > 0$, then $f(aS_1 + b) = aS_1 + b$. If $aS_1 + b \leqslant 0$, then $f(aS_1 + b) = 0$. $f(aS_1 + b)$ then multiplies with another weight, "e", and the product moves to the next line. The ReLU function decides whether the sum impacts the

output by making it 0 or 1. If it is 0, then e*0 will not impact the output. The next step is, to sum up $aS_2 + b$ and $ef(aS_1 + b)$ and go through another ReLU function, multiply by "e", and move to the next input S_3, all the way until all t inputs are in the function, layer by layer.

After that, the resulting f is multiplied by "c" (another weight) and added with "d" (another bias) to create the S_{t+1}, the estimated stock price after the day t based on only past stock prices. Now S_{t+1} becomes the input of the past stock price metric in another neural network in step 2.



We use an RNN for past stock price data because it allows all inputs from different times in history to have the impact of different levels on shaping the output, and the more recent inputs can even have higher impacts than very old ones.

S_{t+1} already looks like the final result we want, but it is only based on historical data of past stock prices. We must also consider the other metrics that refurbish S_{t+1} to reach the final output.

Step 2: NN for all metrics

Now, we have input data for 11 metrics, which have become the features of our traditional neural network, which we will use to estimate future stock prices accurately. We input all of them

simultaneously; they will forward pass to go through several hidden layers, which calculate and combine products to create a final output, the final estimated stock price "y" for the day after t. The inputs are multiplied with each weight in the first layer, and their products are multiplied with each weight in the second layer, and so on. The weights control each input's effect on the outcome. The further we go, the more compressed the products are. We may achieve an accurate and decent future stock prediction as we sum up the products of the last layer because we took many metrics into account and processed them through many layers of weights.

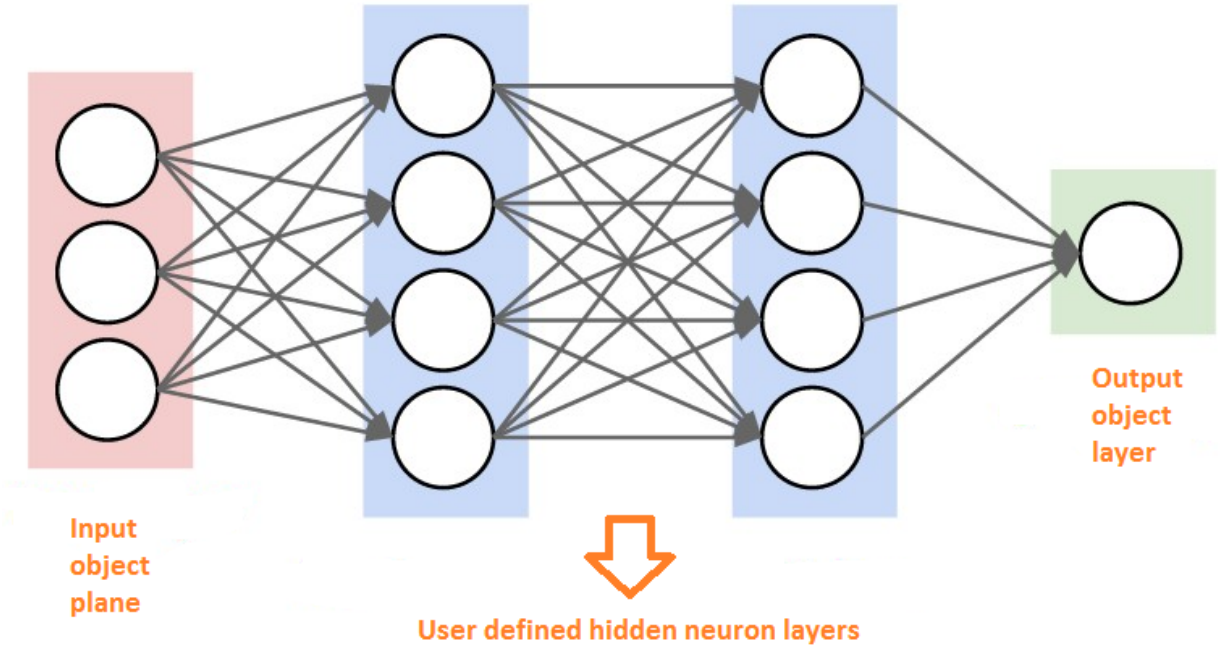


Figure 1. Traditional Convolutional Network Model

The diagram above resembles the real-life situation of my traditional neural network used to predict stock prices. Although it is simpler because it has fewer inputs, weights, and layers, it still delineates the general idea of how different stock metrics become inputs. The network processes them layer-by-layer using adjusted weights, combining the resulting products of the last layer into one output as the estimated future stock price.

We use this neural network because it can learn and improve its accuracy over time by using training data to perform complex tasks that would otherwise require rule-based programming or were only possible for humans to perform.

Step 3: Improving the models

At first, we set weights randomly in the models so the outputs would be inaccurate. To fix this issue, we can calculate the loss of the output (the gap between actual stock price and prediction), backpropagate to adjust weights layer by layer using calculus to reach minimum MSE

(mean square error), train the model several times through backpropagation, and test it using unused inputs.

In the end, we will have a successful neural network that accurately and efficiently predicts future stock prices!