# Introduction to Recurrent Neural Networks(RNNs)

Minor in AI - IIT ROPAR

6th March, 2025

## The Journey of a Financial Analyst - Predicting Stock Prices Using Sequential Data

In the bustling financial district of New York City, Michael, a young and ambitious financial analyst, was obsessed with one thing: predicting stock prices with the highest accuracy possible. He had spent years studying market trends, poring over financial reports, and analyzing past data with traditional statistical models. However, something was missing—his models could not capture the inherent temporal dependencies of stock prices. The patterns seemed random, and no matter how much he tweaked his regression models or used moving averages, they often failed to predict sharp turns in the market.

One evening, while reading about machine learning advancements, he stumbled upon a fascinating concept—**Recurrent Neural Networks (RNNs)**. These networks, unlike traditional models, had the capability to remember past information and use it to make better predictions. Excited by this discovery, Michael decided to delve deeper into the world of RNNs and use them to forecast stock prices more effectively.

## Why Use RNNs for Stock Price Prediction?

Michael realized that stock price prediction required handling sequential dependencies. Unlike traditional models that viewed each day's stock price as independent, RNNs could understand how the stock's price on previous days influenced its future price.

RNNs offered:

- **Sequential Learning**: They processed data step-by-step while maintaining memory of previous values.
- **Context Awareness**: Unlike feedforward networks, RNNs remembered past observations, crucial for financial forecasting.
- **Adaptability**: They could be trained on various sequences—daily, weekly, or even minute-wise stock fluctuations.

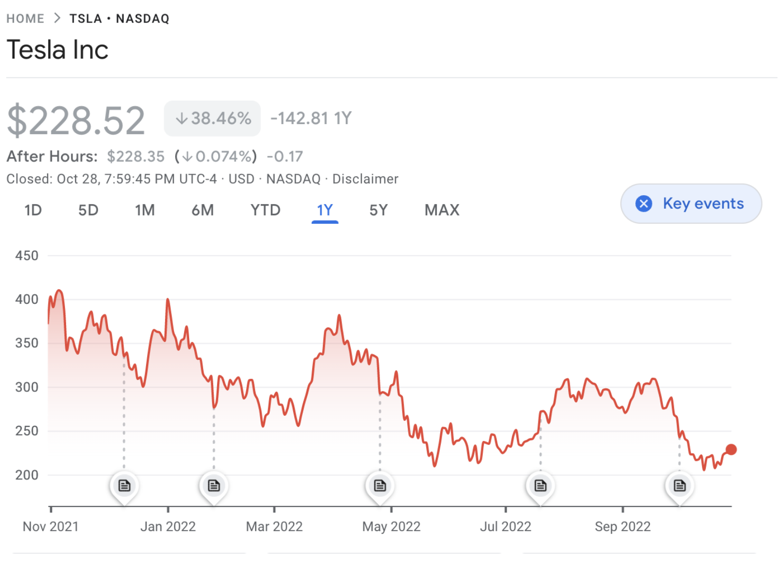
Michael decided to experiment with RNNs and test their effectiveness in real-world financial markets.

## Understanding Time-Series Data

Before implementing an RNN, Michael needed to grasp the key characteristics of time-series data. Stock prices exhibited:

- **Trends**: Long-term upward or downward movements.
- **Seasonality**: Repetitive price movements influenced by economic cycles.
- **Autocorrelation**: Strong influence of past values on future predictions.
- **Noise**: Random fluctuations that made predictions challenging.

He needed a model that could effectively capture trends while filtering out noise, making RNNs a strong candidate.



# Revisiting Neural Networks: The Foundation of RNNs

Before RNNs, Michael had worked with traditional **Artificial Neural Networks (ANNs)**. These were made up of interconnected layers of neurons:

### Structure of a Feedforward Neural Network

A feedforward network consists of:

- **Input Layer**: Accepts stock prices and related financial indicators.
- **Hidden Layers**: Processes data using activation functions (e.g., ReLU, sigmoid).
- **Output Layer**: Produces the final price prediction.

A neuron in an ANN operates as follows:

$$y = f(WX + b) \tag{1}$$

where:

- $y$ is the predicted stock price.
- $W$ is the weight matrix.
- $X$ is the input feature vector.
- $b$ is the bias term.
- $f$ is the activation function.

## Introduction to Recurrent Neural Networks (RNNs)

RNNs provided the missing link by allowing neural networks to retain past information. Instead of treating each stock price as a separate event, RNNs maintained a hidden state that stored information from previous days.

## Understanding the RNN Architecture

### Basic Structure

An RNN consists of:

- **Input layer**: Accepts a sequence of data points (e.g., words in a sentence or stock prices).
- **Hidden layer with loops**: Stores memory of previous computations.
- **Output layer**: Produces the final prediction.

### Mathematical Formulation

At each time step $t$, the RNN takes an input $x_t$, updates its hidden state $h_t$, and produces an output $o_t$.

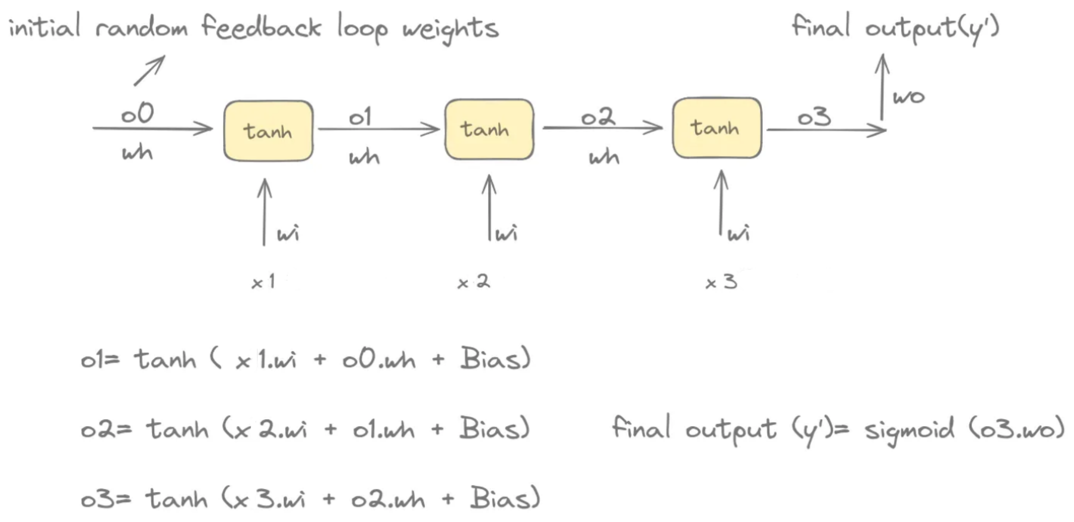
$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \tag{2}$$

$$o_t = W_o h_t \tag{3}$$

Where:

- $h_t$ = Hidden state at time $t$
- $x_t$ = Input at time $t$
- $W_x, W_h, W_o$ = Weight matrices
- $b$ = Bias
- $\tanh$ = Activation function (to keep values between -1 and 1)

## Visual Representation


initial random feedback loop weights
final output(y')
o0
wh
tanh
o1
wh
tanh
o2
wh
tanh
o3
wo
wi
wi
wi
x 1
x 2
x 3
o1= tanh ( x 1.wi + o0.wh + Bias)
o2= tanh (x 2.wi + o1.wh + Bias)
o3= tanh (x 3.wi + o2.wh + Bias)
final output (y')= sigmoid (o3.wo)

This structure represents:

- A sequence of inputs $x_1, x_2, x_3$.
- A series of tanh activation functions storing past information.
- A final output computed using a sigmoid function.

## Step-by-Step Breakdown of RNN Computation

Let's consider a simple 3-step RNN :

### At $t = 1$

$$o_1 = \tanh(x_1 W_i + o_0 W_h + \text{Bias}) \tag{4}$$

- The first input $x_1$ is multiplied by weight $W_i$.
- The previous hidden state $o_0$ (initially 0) is multiplied by $W_h$.
- The sum is passed through a tanh activation function.

### At $t = 2$

$$o_2 = \tanh(x_2 W_i + o_1 W_h + \text{Bias}) \tag{5}$$

- The previous output $o_1$ influences the current state.

### At $t = 3$

$$o_3 = \tanh(x_3 W_i + o_2 W_h + \text{Bias}) \tag{6}$$

- The final output is calculated using:

$$y' = \sigma(o_3 W_o) \tag{7}$$

- Here, $\sigma$ is the sigmoid activation function, mapping the output to a probability (useful in classification tasks).

## Types of Recurrent Neural Networks

Recurrent Neural Networks (RNNs) come in various architectures, each suited for different types of sequence-processing tasks.The main types of RNNs include One-to-One, One-to-Many, Many-to-One, and Many-to-Many architectures.

### 1. One-to-One (Vanilla RNN)

This is the simplest form of an RNN, resembling a traditional neural network but with recurrent connections. It takes a single input and produces a single output, making it functionally similar to a standard feedforward neural network.

### 2. One-to-Many RNN

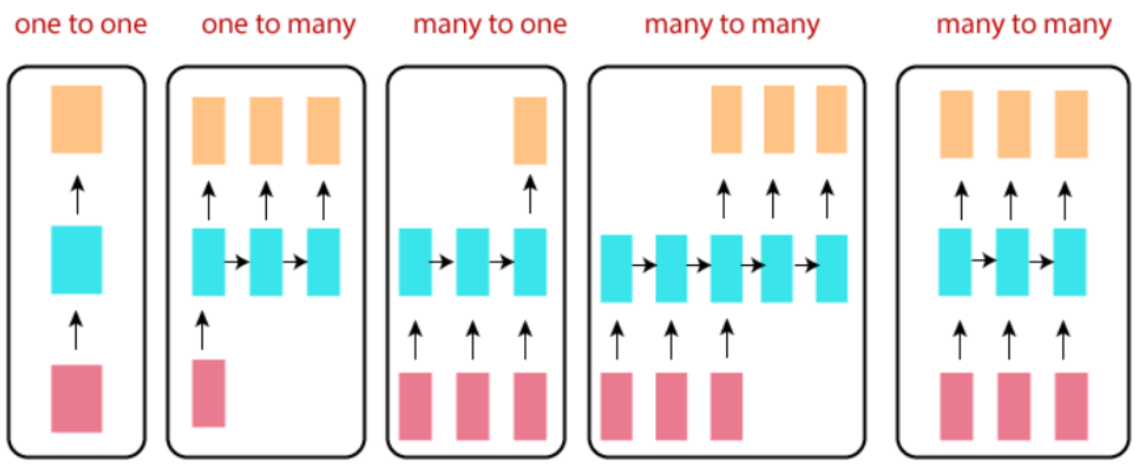
In this type of RNN, one input produces multiple outputs over time. This structure is particularly useful for applications where a single data point is expanded into a sequence. This architecture is widely applied in music generation, image captioning, etc.

### 3. Many-to-One RNN

This architecture takes multiple inputs and produces a single output. It is particularly useful for tasks where an entire sequence of data needs to be analyzed before generating a final prediction. Many-to-One RNNs are commonly used in sentiment analysis and spam detection.

### 4. Many-to-Many RNN

This type of RNN processes input and output sequences of the same length, where each input has a corresponding output at each time step. It is particularly suitable for sequence labeling tasks, where each element in the input sequence is assigned a label. Common applications include Named Entity Recognition (NER), Part-of-Speech (POS) tagging and time-series forecasting.



## Backpropagation Through Time (BPTT) and Training Challenges

Training RNNs requires a process called **Backpropagation Through Time (BPTT)**.

### Steps in BPTT

1. **Forward Pass**: Computes predictions for all time steps.
2. **Loss Calculation**: Compares predictions with actual stock prices.
3. **Backward Pass**: Computes gradients for each time step using the chain rule.
4. **Weight Updates**: Adjusts weights using gradient descent.

## Key Takeaways

Recurrent Neural Networks (RNNs) are essential for processing sequential data, as they retain information from previous inputs. Unlike traditional neural networks, RNNs introduce feedback loops that allow information to persist across time steps, making them useful for tasks involving temporal dependencies.

- **RNNs are designed for sequential data**: Unlike feedforward networks, RNNs remember past information, making them ideal for processing sequences like text, speech, and time-series data.
- **The hidden state enables memory retention**: At each time step, the hidden state stores past computations and influences the next state, capturing dependencies between elements in the sequence.
- **RNNs follow a step-by-step computation**: Each hidden state is updated using the input at the current time step, the previous hidden state, and learned weights.
- **Types of RNNs vary based on input-output structures**:
  - **One-to-One (Vanilla RNN)**: Basic model with single input and output.
  - **One-to-Many**: A single input generates a sequence (e.g., music generation).
  - **Many-to-One**: A sequence is processed to produce a single output (e.g., sentiment analysis).
  - **Many-to-Many**: Sequences of varying lengths are processed (e.g., machine translation).
- **Backpropagation Through Time (BPTT) is used for training**: RNNs are trained using BPTT, which unrolls the network in time and applies backpropagation to update weights.

His journey was a testament to the power of deep learning, and he envisioned a future where AI would revolutionize financial decision-making.