# ARTIFICIAL NEURAL NETWORKS - WEEK 9

## Long Short-Term Memory (LSTM)


Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

# CONTENTS

1. Introduction

2. LSTM Cell and Gates Logic

3. Case Study: Ethereum Price Prediction

4. Resources

# LONG SHORT-TERM MEMORY (LSTM)

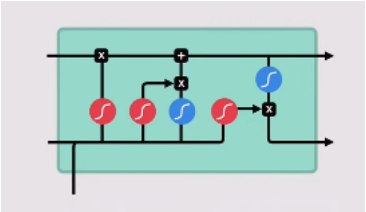
## What is LSTM?

LSTM is a type of RNN architecture. It's designed to overcome the limitations of traditional RNNs in capturing and remembering long-term dependencies in sequential data.

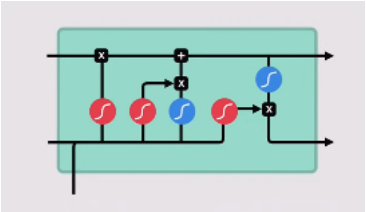
In LSTM networks, each cell has a more complex structure compared to a basic RNN unit. It contains a memory cell that can maintain information over time, as well as various gates that regulate the flow of information into and out of the cell. These gates include the input gate, forget gate, and output gate.

1. `Input gate:` it determines how much new information is allowed into the memory cell.
2. `Forget gate:` it decides which information to discard from the memory cell.
3. `Output gate:` it controls the extent to which the information in the memory cell is used to compute the output of the LSTM unit.

# LSTM ARCHITECTURE



# LSTM ARCHITECTURE



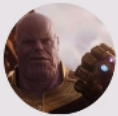
# FORGOT GATE

**Why Forgot Gate is required?**

1. `Long-Term Dependencies:` In sequences of data, there are often long-term dependencies where certain information becomes less relevant as the sequence progresses. The forget gate allows the LSTM to selectively discard information from the past that is no longer relevant for the current context.
2. `Addressing Vanishing Gradient:` Traditional RNNs often suffer from the vanishing gradient problem, where gradients diminish exponentially as they propagate back through time during training. By selectively forgetting irrelevant information through the forget gate, LSTMs mitigate the vanishing gradient problem by allowing the network to focus on relevant information, thereby preserving the gradients that contribute to learning meaningful patterns in the data.
3. `Dynamic Memory Management::` The forget gate enables LSTMs to dynamically manage their memory by updating and maintaining only the most relevant information in the memory cell while discarding less important information.

# FORGOT GATE

**Customers Review** 2,491



**Thanos**

September 2018

Verified Purchase

**Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!**


**A Box of Cereal**

**$3.99**

# FORGOT GATE

**Customers Review** 2,491

**Thanos**

September 2018

Verified Purchase

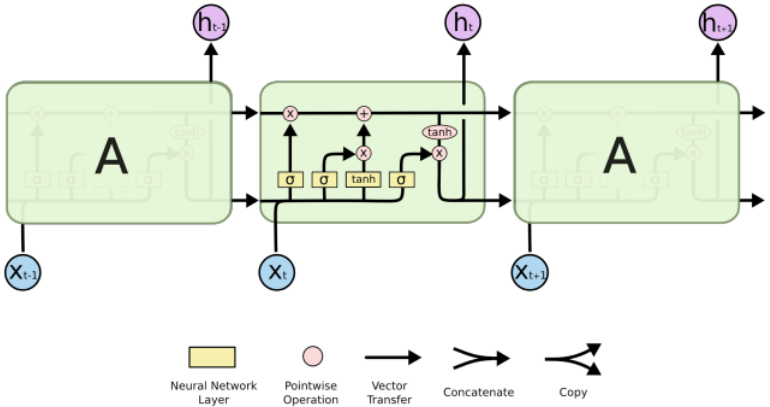
**Amazing!** This box of cereal gave me a **perfectly balanced breakfast**, as all things should be. I only ate half of it but **will definitely be buying again!**


**A Box of Cereal**

**$3.99**

# LSTM LAYERS

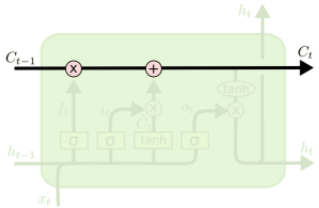
An LSTM is composed of four different layers.

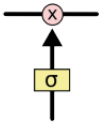


# LSTM CELL STATE OR MEMORY

The cell state is kind of like a conveyor belt.

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.



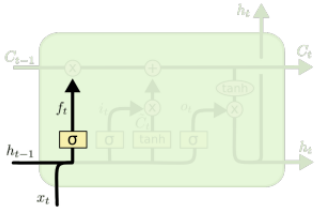

Gates are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

The sigmoid layer outputs numbers between `zero and one`, describing how much of each component should be let through. A value of `zero means let nothing through`, while a value of `one means let everything through`.

# LSTM FORGET GATE

Should we continue to remember this bit of information or not?

$f_t = 1$ represents `completely keep this` while a 0 represents `completely get rid of this`.



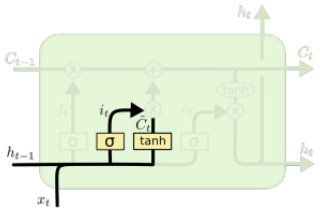

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

**Example:** we want to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

# LSTM INPUT GATE

Should we update this bit of information or not?

If so, with what?



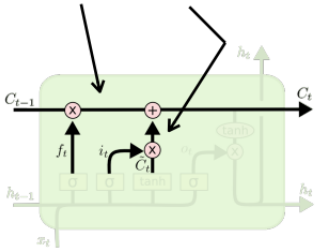

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

**Example:** in the example of our language model, we want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

# LSTM MEMORY UPDATE

- Forget that + memorize this




$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

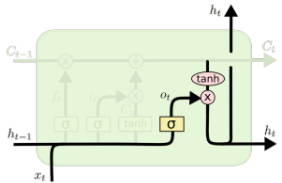
**Example:** in the case of the language model, this is where we will actually drop the information about the old subject's gender and add the new one.

# LSTM OUTPUT GATE

What shall we output given the context?

The output is a function of the cell state ($C_t$). A `tanh` pushes the cell state values towards $+1$ or $-1$.

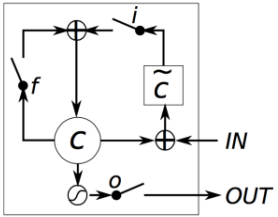
A `Perceptron` selects what to output based on the current input $x_t$ and the previous output $h_{t-1}$, by multiplying its output $o_t$ and $tanh(c_t)$.




$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

**Example:** in the case of the language model, given that we just saw a new subject, this cell might decide how to conjugate a verb, based on its knowledge of the subject, whether it is plural or singular.

# THE LSTM BUILDING BLOCKS




- There are three gates (i,f,o) controlled by NNs (e.g., Perceptrons) weighing the inputs and the recurrent outputs.

- Given an input IN, we compute a new state $\bar{C}$ that can:

1. be ignored ($i = 0$)
2. replace the previous state ($f = 1$ and $i = 1$).
3. be used to compute a new state $C + \bar{C}$.

- Given a state $C$, the network outputs OUT ($o = 1$) or not ($o = 0$).

# YAHOO FINANCE DATASET

The Yahoo Finance dataset is a collection of financial data on various assets traded in the financial markets. This dataset provides historical pricing information and range of financial metrics including cryptocurrencies.

In python, the Yahoo Finance dataset is accessed using the `yfinance` library. You will need to install it first using `pip install yfinance`.

**Downloading Ethereum Data from Yahoo Finance**: The line of code below uses the download function from the `yfinance` library to download historical price data of ETH against the USD currency pair. The data is downloaded for the time period from January 1, 2018, to January 1, 2024.

```
1 eth_data = yf.download('ETH-USD', start='2018-01-01', end='2024-01-01')
```

# YAHOO FINANCE DATASET

The dataset gives valuable information to explore Ethereum's market history. Each row represent a day. The fields (features) represent the ETH-USD price of the day including the highest, lowest and closing ETH-USD price for a day represented as high, low and close respectively.

| Date | # Open | # High | # Low | # Close | # Adj Close | # Volume |
|---|---|---|---|---|---|---|
| 2018-01-01 | 755.7570190429688 | 782.5300292968675 | 742.0040283203125 | 772.6409912109375 | 772.6409912109375 | 2595760128 |
| 2018-01-02 | 772.3460083007812 | 914.8300170898438 | 772.3460083007812 | 884.4439697265625 | 884.4439697265625 | 5783349760 |
| 2018-01-03 | 886.0 | 974.4710083007812 | 868.4509887695312 | 962.719970703125 | 962.719970703125 | 5093159936 |
| 2018-01-04 | 961.7130126953125 | 1045.0799560546875 | 946.0859985351562 | 980.9219970703125 | 980.9219970703125 | 6502859776 |
| 2018-01-05 | 975.75 | 1075.3900146484375 | 956.3250122070312 | 997.719970703125 | 997.719970703125 | 6683149824 |
| 2018-01-06 | 995.1539916992188 | 1060.7099609375 | 994.6220092773438 | 1041.6800537109375 | 1041.6800537109375 | 4662219776 |

# PREPROCESSING

The preprocessing prepare the Ethereum price data by performing two essential tasks. First, it extracts the closing prices of Ethereum, focusing specifically on this key metric for analysis. Then, utilizing the extracted data, using min-max scaling to normalize the closing prices, ensuring that they fall within the range of [0, 1].

```
def preprocessing(self):
        # Processing the data
        data = self.eth_data['Close'].values.reshape(-1, 1)
        self.scaler = MinMaxScaler(feature_range=(0,1))
        self.scaled_data = self.scaler.fit_transform(data)
```

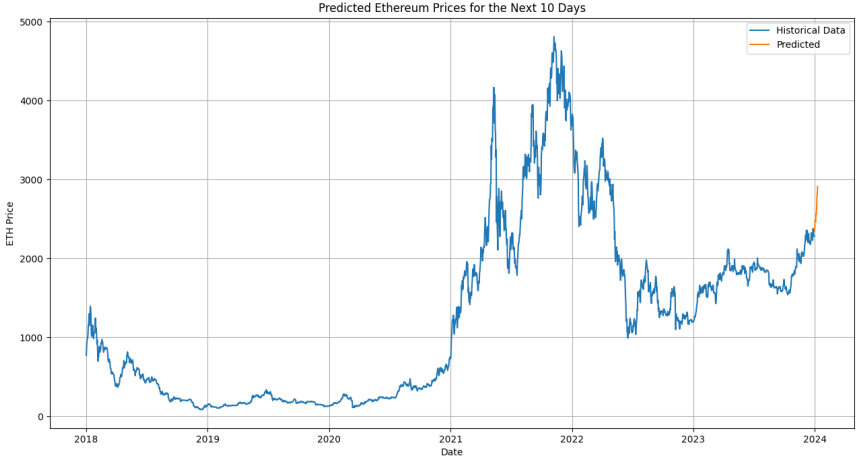
Next, TimeseriesGenerator is used to construct generators for both training and testing datasets. Each generator produce sequences of input-output pairs, where both the input and output sequences are derived from the scaled Ethereum price data. The length parameter specifies the length of each input sequence, while the batch_size parameter controls the number of samples per batch.

# MODEL

```
 Layer (type)   Output Shape    Param
=================================
 lstm (LSTM)    multiple        10400
 lstm_1 (LSTM)  multiple        20200
 dense (Dense)  multiple        51
=================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
```

For LSTM layers, the parameter count is determined by the number of units (50 in each LSTM layer) and the number of input features (1 in this case). The calculation involves multiple factors, including weights, biases, and gates within the LSTM cells. The Dense layer has 51 parameters, which includes the weight matrix of size (50, 1) and the bias vector of size (1,). The extra parameter accounts for the bias term.

# ETH PRICE PREDICTION FOR NEXT 10 DAYS



# RESOURCES

To download the source codes used in the previous slides, follow the link:

▸ Download Source Codes

Import the codes into your preferred development environment, such as Visual Studio Code (VS Code), to practice and explore further.

To learn programming in Python, follow my comprehensive 15-week Programming in Python course at:

▸ Programming in Python