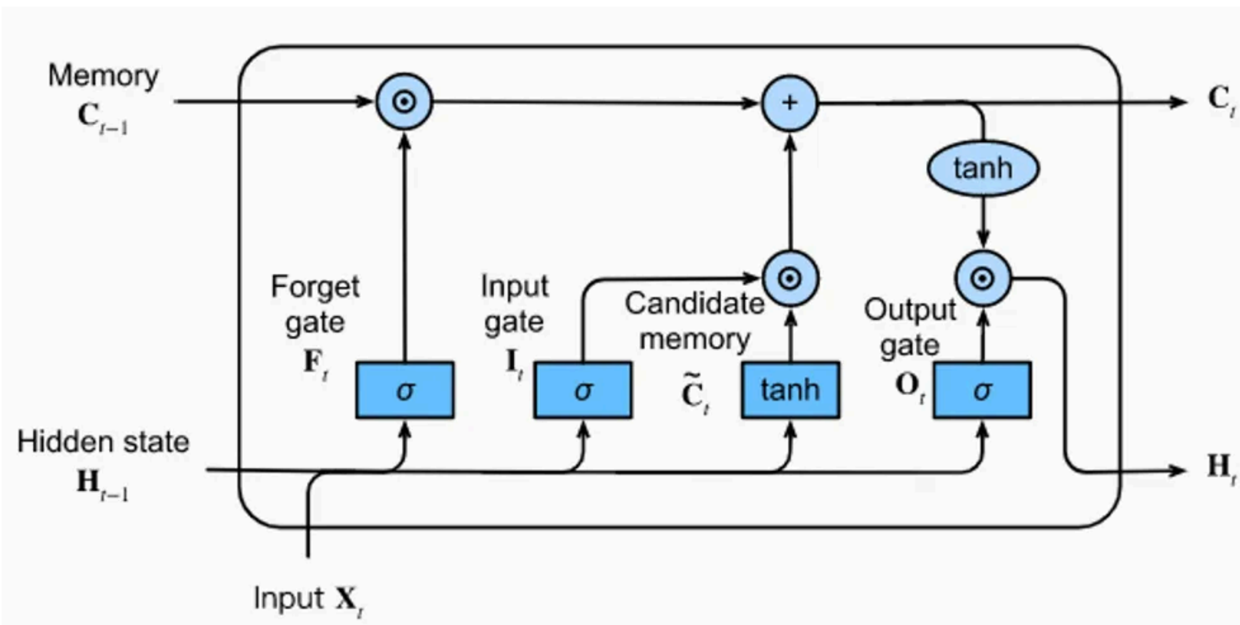
LSTM

About the Model: LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) capable of learning long-term dependencies. It is designed to overcome the limitations of vanilla RNNs, which suffer from vanishing and exploding gradients when handling long sequences.



Architecture of a LSTM Unit

Core Concepts

- **Memory Cells**: The key innovation of LSTMs is the introduction of a memory cell that can maintain information in memory for long periods.

- **Gates**: Each LSTM unit includes:
  - **Forget Gate**: Decides what information to discard.
  - **Input Gate**: Decides what information to store in memory.
  - **Output Gate**: Controls how much of the cell state to output.

Working Mechanism

At each time step:

- LSTM takes an input vector x_t, the previous hidden state $h_{(t-1)}$, and the previous cell state $c_{(t-1)}$.
- Using learned weights, it computes new values for gates and updates the cell and hidden state.
- This allows LSTMs to **retain useful past information** and **forget irrelevant details** dynamically.

Key Features

- **Handles long sequence data** better than traditional RNNs.
- **Prevents vanishing gradient problem** during training via gate control.
- Supports **multi-layer stacking** for complex sequence modeling.
- Widely used in both **univariate and multivariate** time series problems.

Use Cases

LSTMs are highly versatile and appear in numerous domains:

Domain	Applications
NLP	Language modeling, text generation
Finance	Stock price prediction, anomaly detection
IoT / Time Series	Forecasting sensor data, energy usage
Healthcare	Patient monitoring, ECG signal analysis
Audio / Speech	Speech recognition, music generation

Implementation Summary

This notebook demonstrates the basic implementation of LSTM for binary classification using Keras.

Dataset Preprocessing

- Loaded data and split into x_train, y_train, x_test, y_test
- Checked shape and reshaped inputs into 3D format (samples, timesteps, features) as required by LSTM

Model Architecture

python

CopyEdit

```
model = Sequential()
model.add(LSTM(128, input_shape=(timesteps, features), return_sequences=True,
unroll=True))
model.add(LSTM(64, unroll=True))
model.add(Dense(1, activation='sigmoid'))
```

- Used unroll=True to ensure compatibility in environments where cuDNN is unavailable
- Final layer uses **sigmoid activation** for binary classification

Training

- Trained for 5 epochs with batch_size=64
- Included validation using x_test, y_test

GPU Compatibility

- Initial issues with CudnnRNN were resolved by:
    - Using CPU-compatible LSTM (unroll=True)
    - Ensuring data is reshaped properly

- Verifying GPU access via tf.config.list_physical_devices('GPU')

Conclusion

- LSTM networks provide a powerful approach for modeling sequential data.
- This notebook successfully implemented a simple 2-layer LSTM for binary classification.
- cuDNN compatibility is critical for leveraging GPU acceleration; alternatives like unrolling or CuDNNLSTM are effective workarounds.
- Despite the GPU setup challenges, the model runs efficiently on CPU with minimal performance trade-off for small-scale data.

References

1. Image Source: https://d2l.ai/chapter_recurrent-modern/lstm.html
2. https://notesonai.com/lstm
3. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c