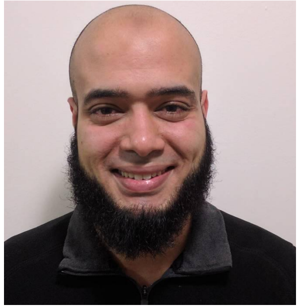
Machine Learning
LSTM

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Long **Short-Term** Memory (LSTM) Networks

- One special kind of RNNs introduced by Hochreiter and Schmidhuber in 1997
- Goal: to address the vanishing gradient problem in RNNs in **long sequences**
- LSTM key goal is to create a **smarter state** that allows longer flow of information in the feedforward and backpropagation
  - People interpret that as being able to remember for a longer time
  - We call it the cell state
- To do so, LSTM introduces the concept of **gates** that will help it **control** the flow of information: 3 gates
  - What to **forget** from the from the last cell state?
  - What **new information** we’re going to store in the cell state?
  - What to select for the **output**?

Gates Concept

- Gates are used to **control the flow** of information through the network.
    - **Control**: what to pass and what to discard
    - As a result, we also control the gradient backpropagation!
- **Implementation**
    - Typically implemented using **sigmoid** activation functions
        - These values are used to **scale** the amount of information passed through.
        - A value close to 0 means "let nothing through,"
        - A value close to 1 means "let everything through."
    - Example: input [10, 30, 8] and sigmoids [0, 0.5, 1] ⇒ **element wise** multiplication:
        - [10 x 0, 30 x 0.5, 8 x 1] = [0, 15, 8] ⇒ cancel 10, half of 30 and keep all 8
    - We learn **weights** to decide the gates values
- Popular examples:
    - LSTM (forget, input, output gates) - GRUs (update, rest gates)

Gates Roles

- Memory Management:
  - We can’t keep all historical information (RNN fails)
  - We can **select** information to remain over long sequences
- Gradient Flow:
  - As we control the feedforward flow, we control the gradients tii
  - This help mitigate the vanishing and exploding gradient problems
- Modeling Complex Patterns:
  - With the flow control, the network may learn complex dependencies and patterns
- Interpretation: Assume a sigmoid value 0.7
  - Some people interpret it as we **keep** (select) 70% of the value
  - Some people interpret it as we **forget** (throw away) 30% of the value

Creating a Gate

- Assume you want a gate (e.g. to forget something)
- We utilize 2 1-D vectors
    - Your input vector X for this time step
    - Your hidden state (network output) of the last step
- We create a transformation that will **learn** the gate
    - We need 2 weight matrices to transform each vector and add bias
    - Transform and perform element wise addition
- Finally, use sigmoid to generate the learned [0-1] range

```python
def transform(Wx, x_t, Wh, h_t, b):
    # Transform the 2 vectors and do element-wise additio
    return np.dot(Wx, x_t) + np.dot(Wh, h_t) + b

def gate(Wx, x_t, Wh, h_t, b):
    t = transform(Wx, x_t, Wh, h_t, b)
    return sigmoid(t)   # [0-1] for scaling values
```

Creating Gates

- Based on your logic, you can learn many gates
- Below, we prepare 3 gates. Each gate helps learning 2 weight matrices

```
# What to keep from the history
f_t = gate(Wf, x_t, Uf, h_t, bf)
# What to keep from the input
i_t = gate(Wi, x_t, Ui, h_t, bi)
# What to keep from the output
o_t = gate(Wo, x_t, Uo, h_t, bo)
```

LSTM: Accumulation Trick

- Recall RNN: $$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$
- In LSTM, the first key change is we don't just **REPLACE** the old state with the fused state, but we **ACCUMULATE** to it
  - This accumulation helps the gradient flow. Kind of a **weak skip connection** in **resent**
  - Observe: C_t: series of tanh() additions

```
# Code before gates
h_t, C_t = initial_states
for x_t in inputs:
    # non-linear Transformation for input and history (like RNN)
    fused_state = tanh(transform(Wc, x_t, Uc, h_t, bc))

    # ACCUMULATE to the previous history (Cell state)
    C_t = C_t + fused_state
    # Output hidden state
    h_t = tanh(C_t)
```

LSTM: Gates Trick

- We can identify ~3 components in the last code
- We can learn a gate for everyone and let the network control their flows
- The way the author interpreted them:
  - **Forget Gate**: Decides what information should be thrown away from the cell state.
    - Prevent accumulation of irrelevant information to avoid gradients issues (e.g. vanish)
  - **Input Gate**: Decides which values should be updated in the cell state.
    - They call the fused state; the state candidate!
  - **Output Gate**: Determines what part of the cell state should be output at the current timestep.
- Remember; we learn the weights that help the control

LSTM: Gates Trick

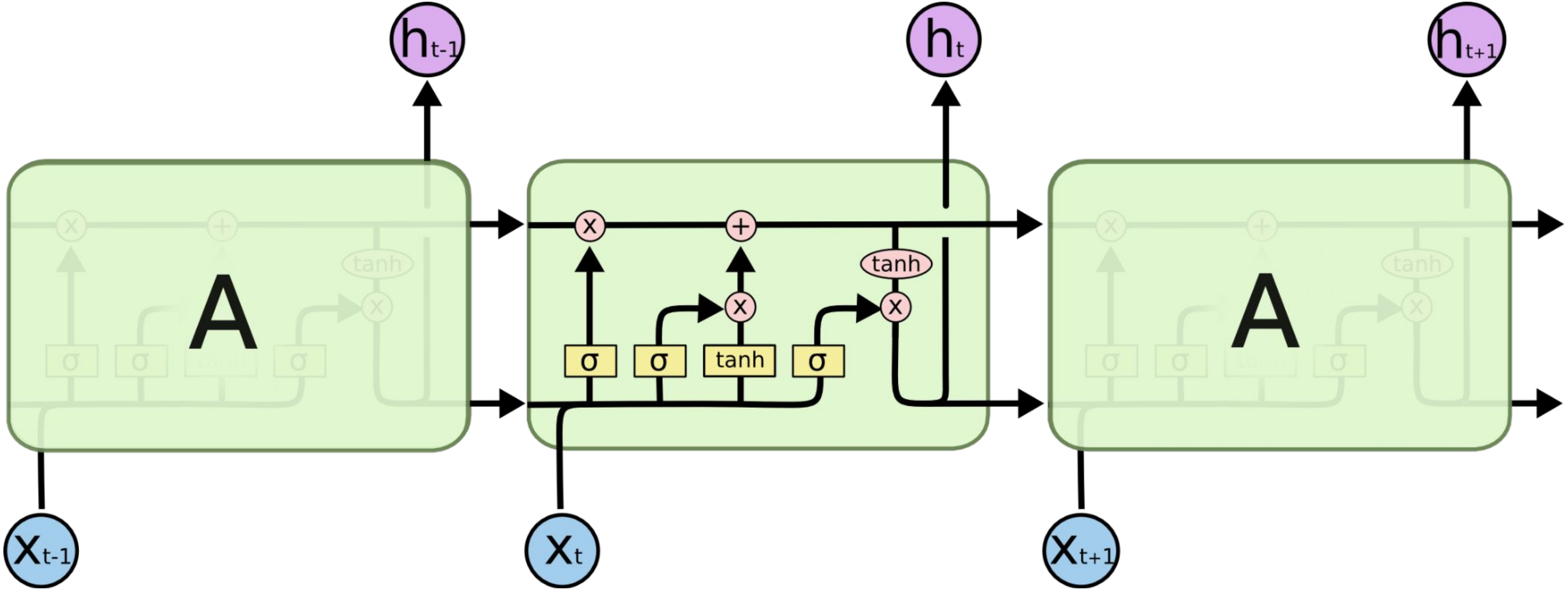
```
h_t, C_t = initial_states
for x_t in inputs:
    # What to keep from the history
    f_t = gate(Wf, x_t, Uf, h_t, bf)
    # What to keep from the input / fused state
    i_t = gate(Wi, x_t, Ui, h_t, bi)
    # What to keep from the output
    o_t = gate(Wo, x_t, Uo, h_t, bo)

    fused_state = tanh(transform(Wc, x_t, Uc, h_t, bc))

    # Select from old state + select from fused state
    C_t = f_t * C_t + i_t * fused_state

    # Select what to output
    h_t = o_t * tanh(C_t)
```

Imagine:

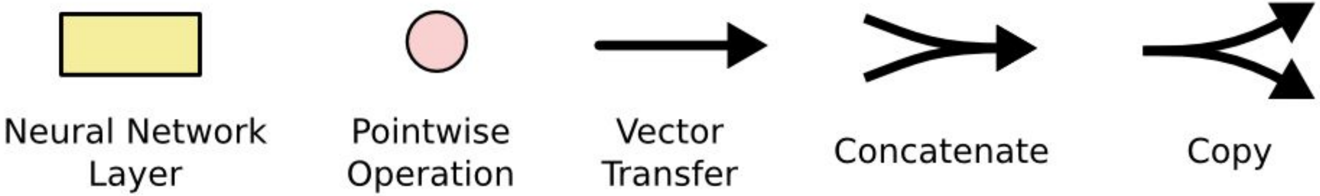
- f_t = 1
- i_t = 0
- This means the network decided to preserve the state and ignore new element (e.g. duplicate frame)

LSTM Overall

- These 2 tricks allows the gradients to flow through the network without undergoing the transformations that typically lead to vanishing or exploding gradients for long sequences.
    - People consider the cell state the long memory and hidden state the short memory!
    - But hidden output state is just tanh(cell state)!
- Common practice: Any input feed to some transformation should have some kind of normalization / activation to have stable network
    - Empirically **tanh** validated across a wide range of sequence modeling tasks
        - Try input scales: **[-1, 1]** and [0, 1]
- In practice, LSTM may work well in range like 100-500 steps
    - Beyond that, gradient will vanish. Depends on the data complexity.
- Note: Gated Recurrent Units (**GRUs**)
    - GRUs simplify the LSTM model with 2 gates only and achieves close performance

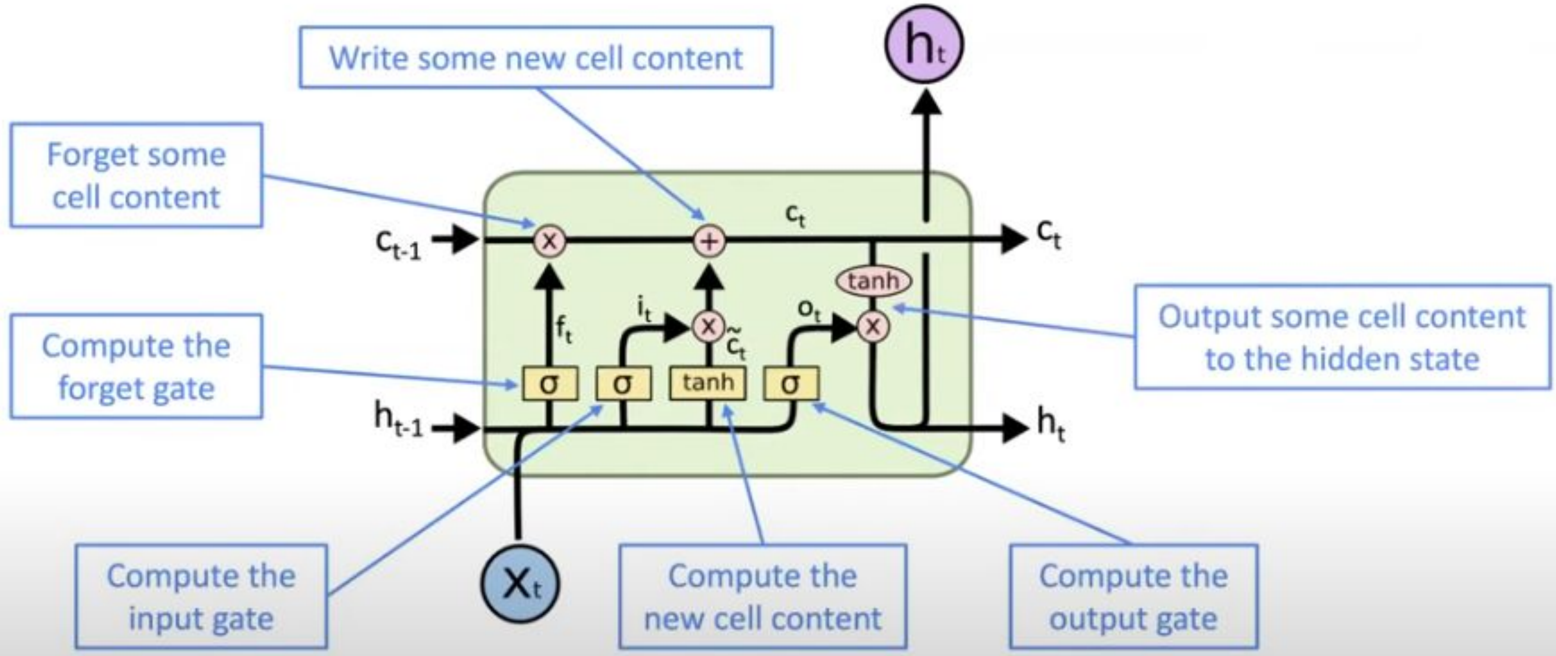


Top horizontal line
is the **cell state**



src

You can think of the LSTM equations visually like this:



src

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Sigmoid function: all gate values are between 0 and 1

Forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

Input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

Output gate: controls what parts of cell are output to hidden state

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

Hidden state: read ("output") some content from the cell

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

Stanford

Stanford CS224N NLP with Deep Learning | Winter 2021 | Lecture 6 - Simple and LSTM RNNs

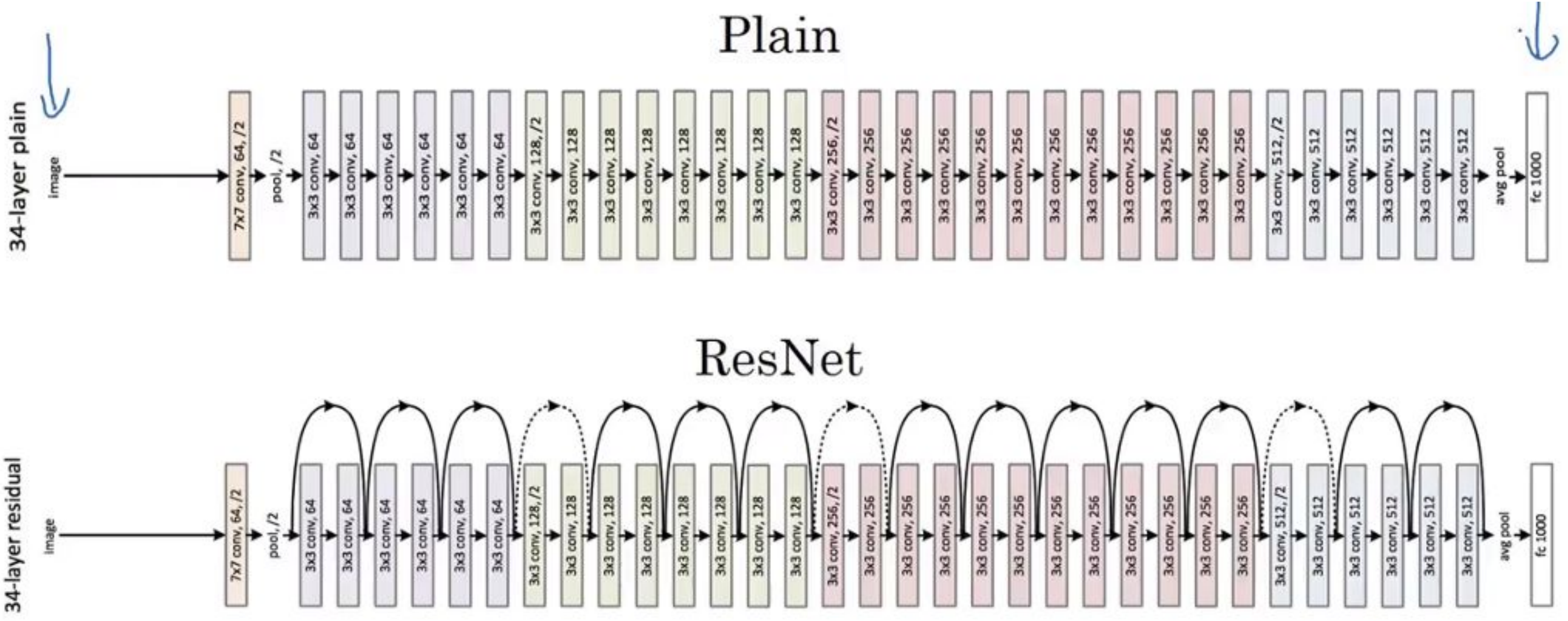
LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks

- Now (2021), other approaches (e.g., Transformers) have become dominant for many tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
    - In WMT 2014, there were 0 neural machine translation systems (!)
    - In WMT 2016, the summary report contains "RNN" 44 times (and these systems won)
    - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf
Source: "Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf

Little about Resnet



src

Relevant Materials

- Understanding LSTM Networks: Article
  - One of the oldest articles in 2015 that helped many people understand LSTM
  - Most of articles just follow it
- StateQuest
- Lecture from Stanford
- There are LSTM variants, e.g. B-LSTM (B for Bidirectional)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."