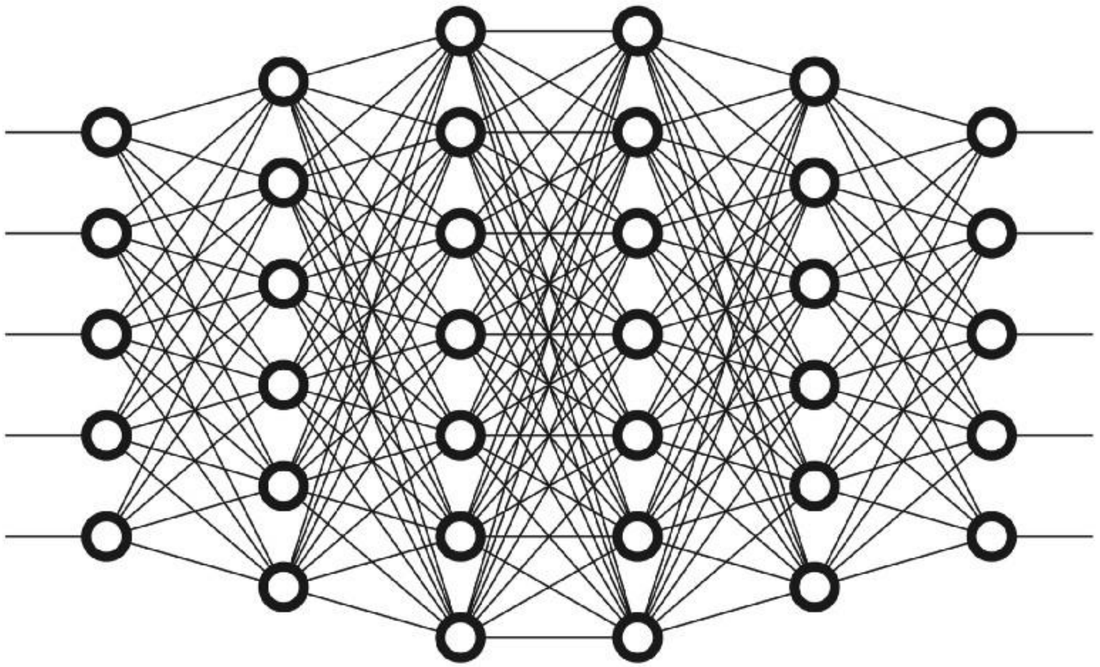
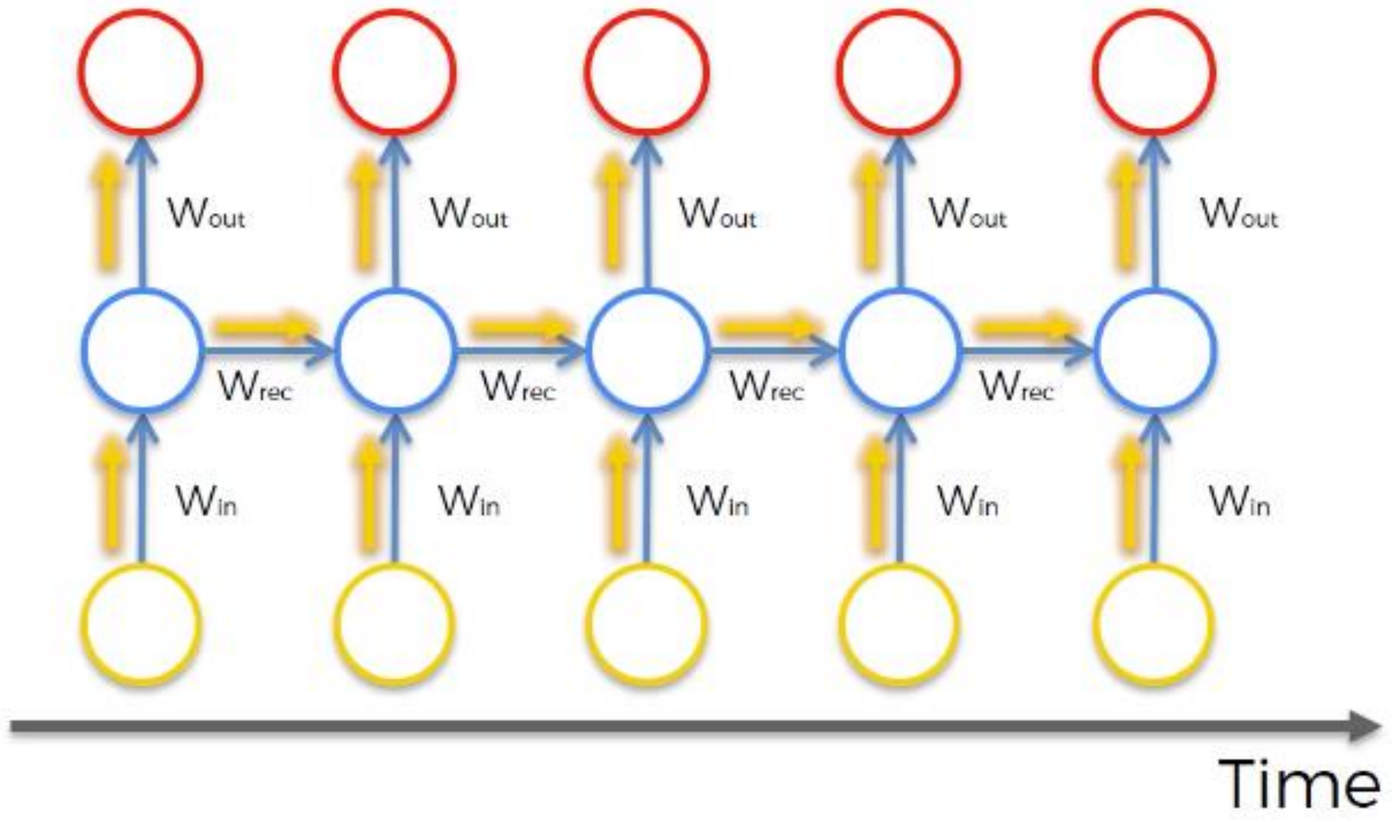
TEXT MINING TUTORIAL #9



Instructor: Prof. Hsing-Kuo Pao
TA: Zolnamar Dorjsembe (Zola)

Date: Apr 23, 2024

Are there any problems with RNN? What do you think?

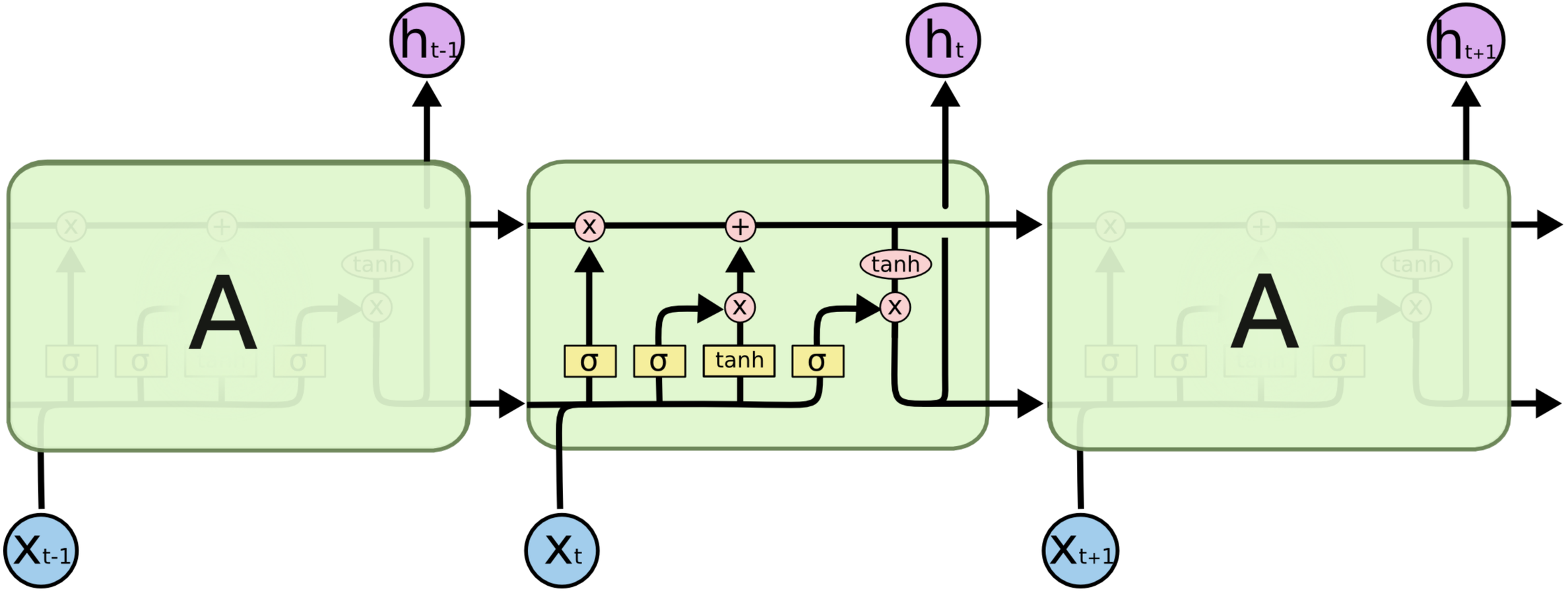


(Source: nickmccullum.com)

The problems with RNN

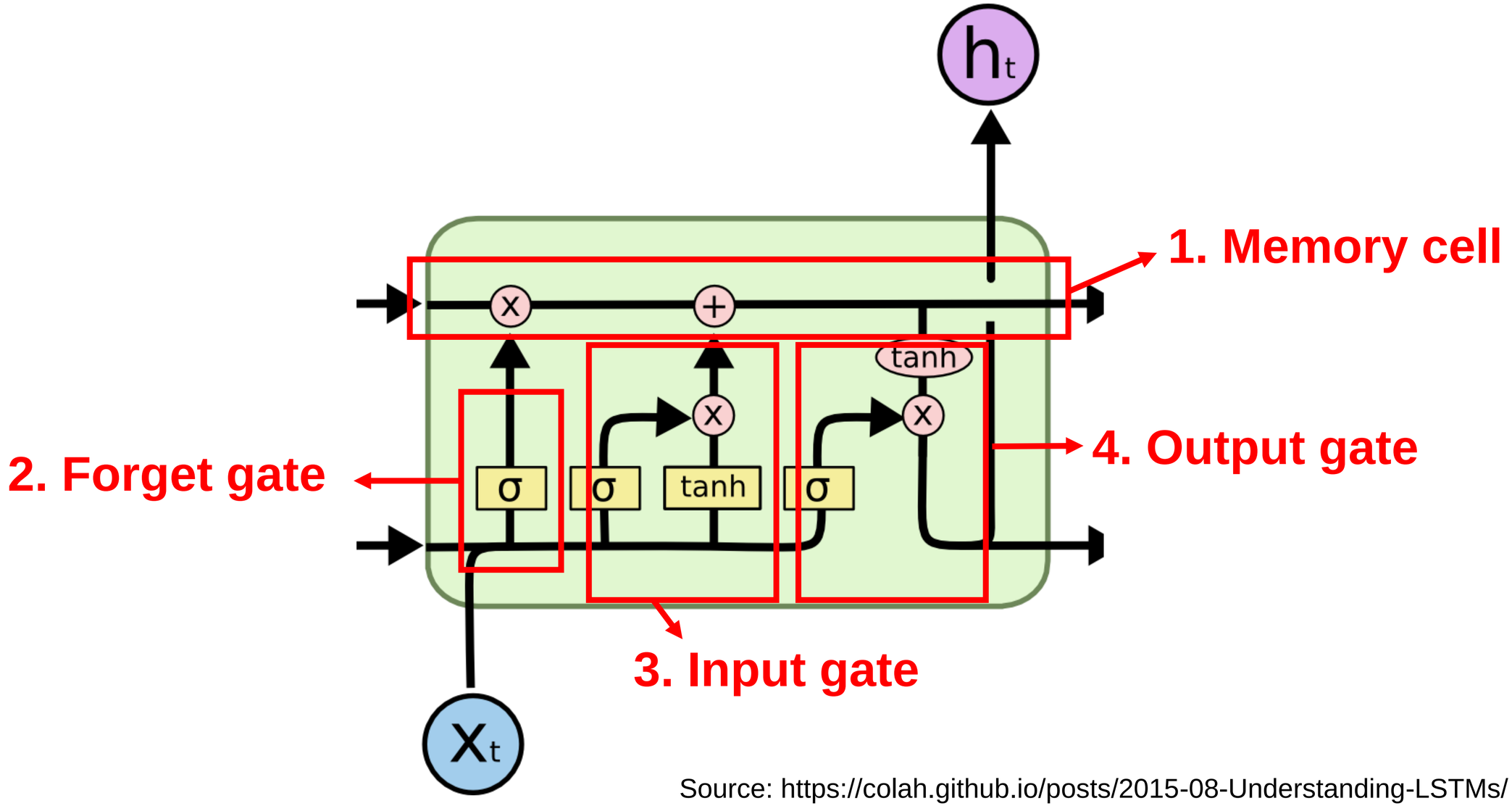
- Vanishing Gradient Problem
- Exploding Gradient Problem (Gradient clipping)
- Long term dependency of words
- Unidirectional in RNN

SOLUTUION: LONG SHORT-TERM MEMORY (LSTM)



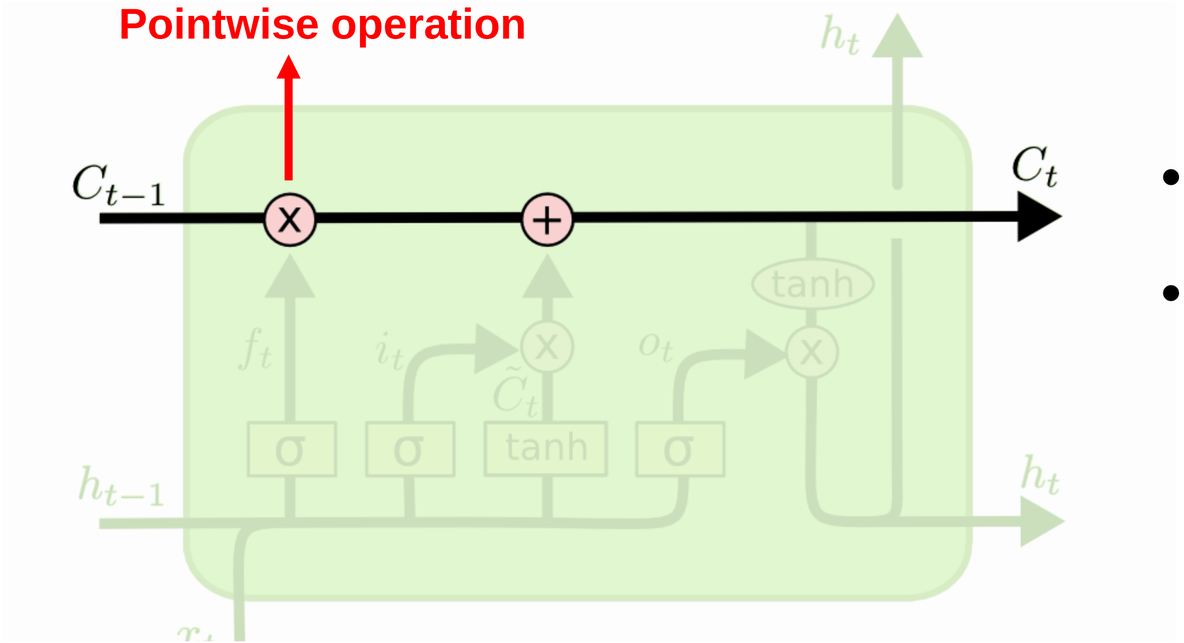
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

SOLUTUION: LONG SHORT-TERM MEMORY (LSTM)



Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

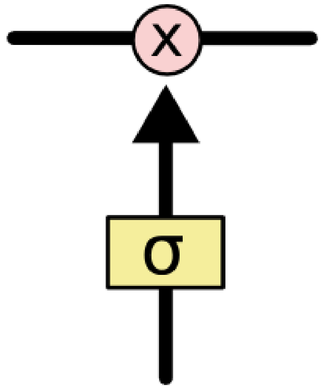
LSTM: 1. MEMORY CELL

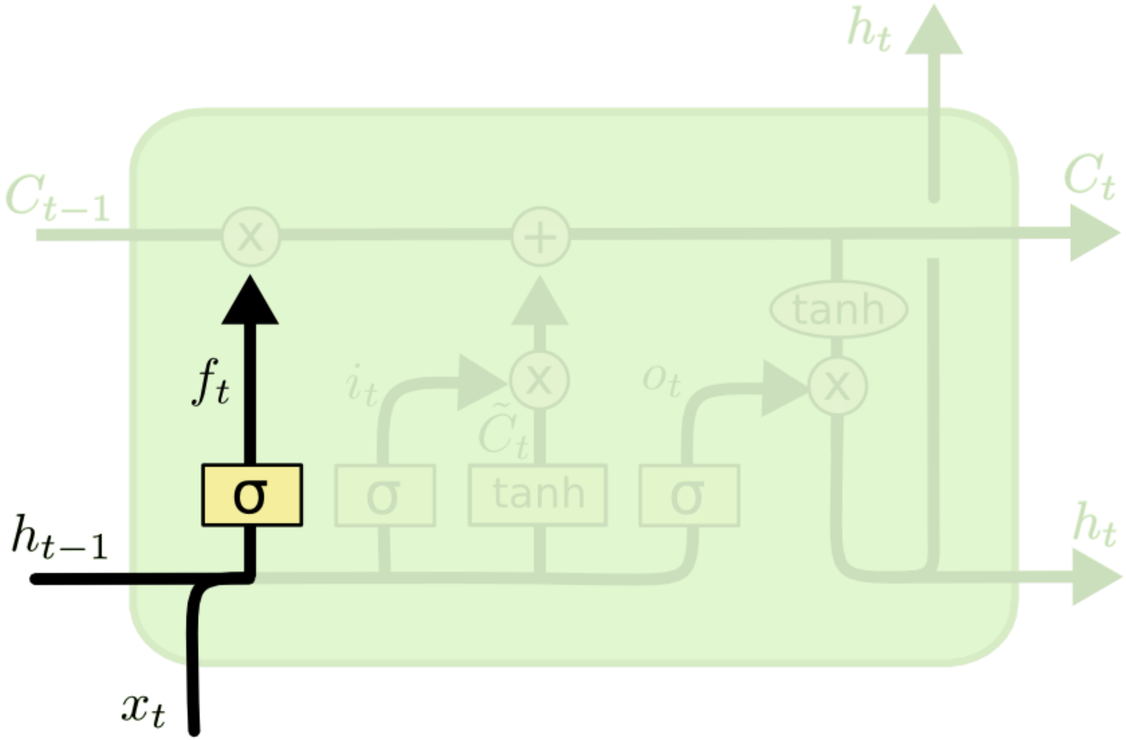


- Forget some information
- Add new information

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: FORGET GATE

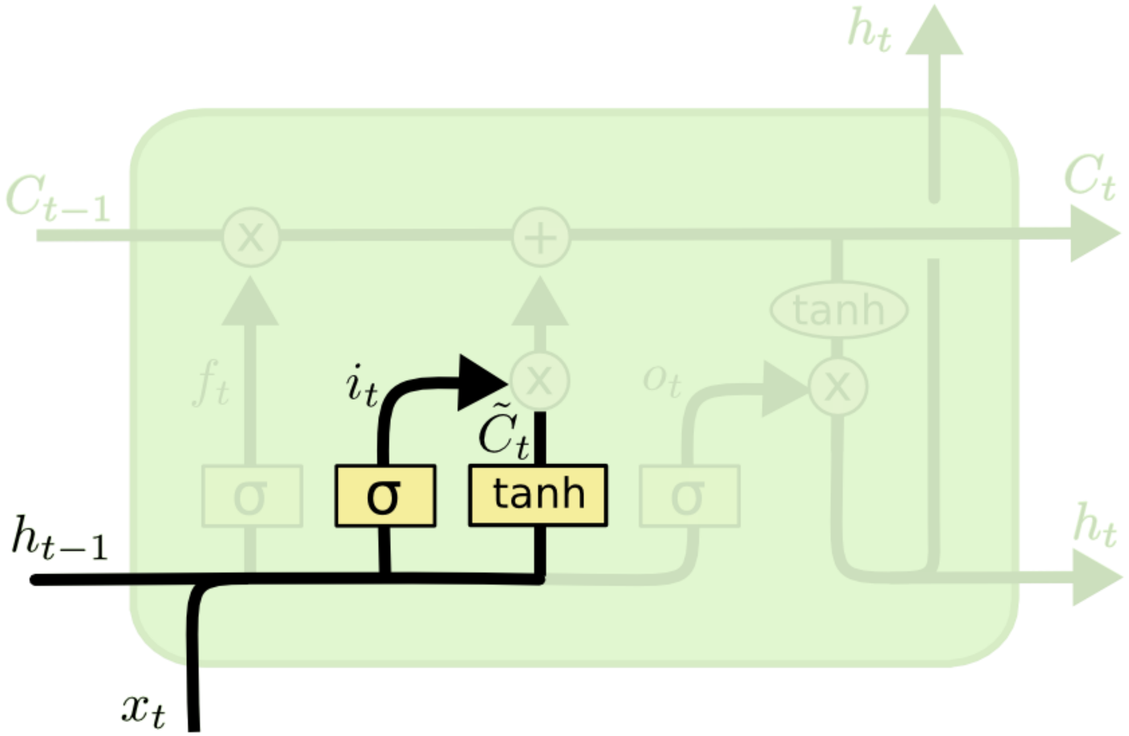




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: INPUT GATE

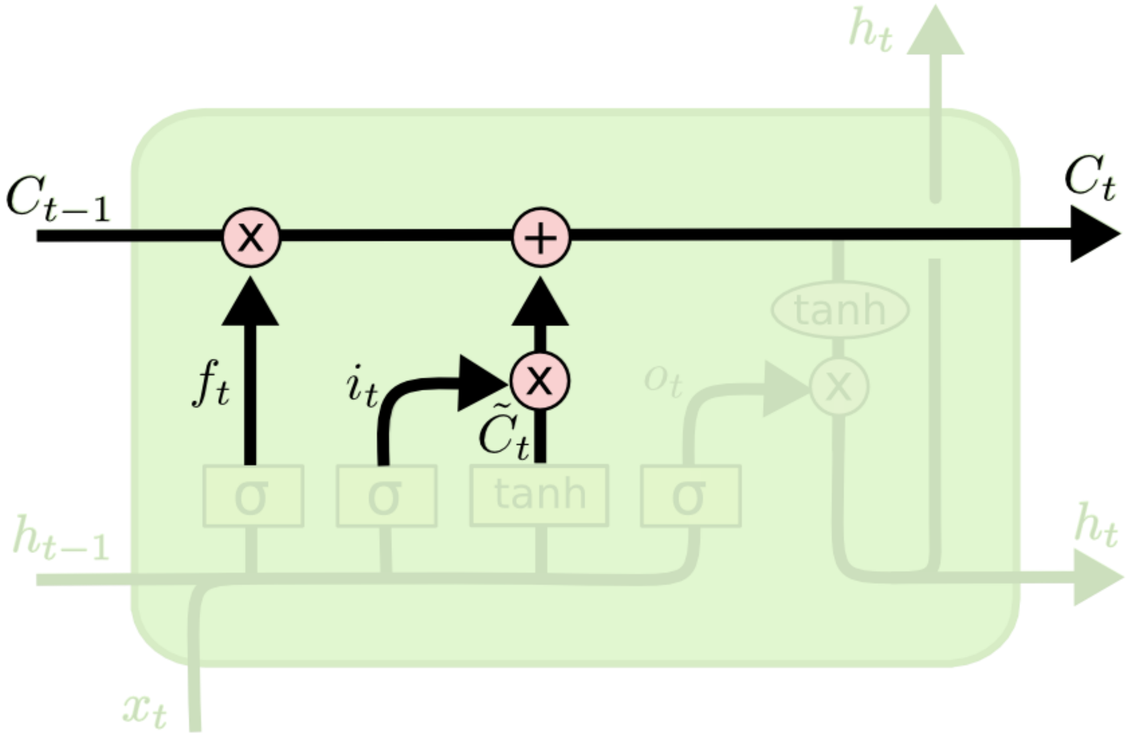


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

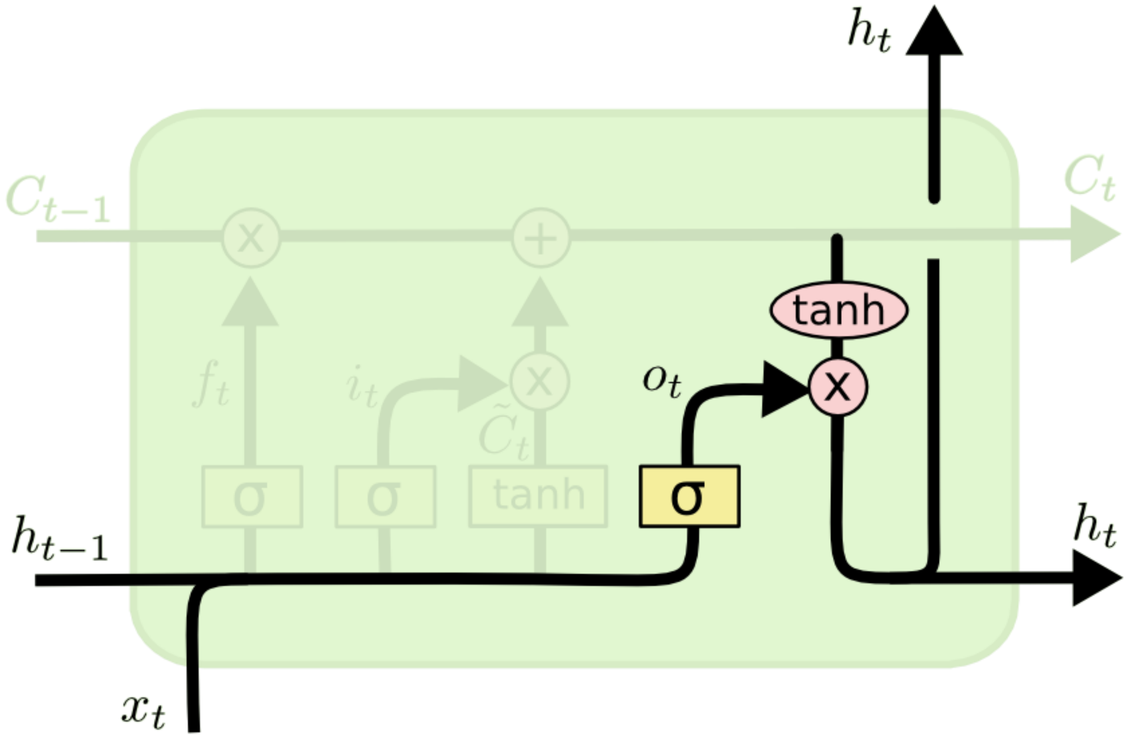
LSTM: INPUT GATE



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

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: OUTPUT GATE



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

$$h_t = o_t * \tanh\left(C_t\right)$$

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LET’S BUILD SIMPLE LSTM IN PYTORCH

SIMPLE ONE-LAYER LSTM

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Text preprocessing
text = "Tell me and I forget, teach me and I may remember, involve me and I learn"
words = text.lower().replace(',', '').split()
vocab = {word: i for i, word in enumerate(set(words))}
vocab_size = len(vocab)

# Prepare data for the model
inputs = []
targets = []
context_size = 3

for i in range(len(words) - context_size):
    input_idx = [vocab[words[j]] for j in range(i, i + context_size)]
    target_idx = vocab[words[i + context_size]]
    inputs.append(input_idx)
    targets.append(target_idx)

inputs_tensor = torch.tensor(inputs, dtype=torch.long)
targets_tensor = torch.tensor(targets, dtype=torch.long)

# Create dataset and DataLoader
```

SIMPLE ONE-LAYER LSTM

```
26 # Create dataset and DataLoader
27 dataset = TensorDataset(inputs_tensor, targets_tensor)
28 dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
29
30 # Define an LSTM model
31 class SimpleLSTM(nn.Module):
32     def __init__(self, vocab_size, embedding_dim, hidden_dim):
33         super(SimpleLSTM, self).__init__()
34         self.embeddings = nn.Embedding(vocab_size, embedding_dim)
35         # LSTM layer, unidirectional and one layer
36         self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
37         # Output layer
38         self.fc = nn.Linear(hidden_dim, vocab_size)
39
40     def forward(self, inputs):
41         # Input shape: (batch_size, sequence_length)
42         embeds = self.embeddings(inputs)  # (batch_size, sequence_length, embedding_dim)
43         lstm_out, (h_n, c_n) = self.lstm(embeds)     # lstm_out: (batch_size, sequence_length, hidden_dim)
44         # We use the last LSTM output as the representation of the sequence
45         final_output = lstm_out[:, -1, :] # (batch_size, hidden_dim)
46         out = self.fc(final_output)       # (batch_size, vocab_size)
47         return out
```

SIMPLE ONE-LAYER LSTM

```
# Model parameters
embedding_dim = 10
hidden_dim = 20

# Initialize the LSTM model, loss function, and optimizer
model = SimpleLSTM(vocab_size, embedding_dim, hidden_dim)
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
epochs = 300
for epoch in range(epochs):
    total_loss = 0
    for context, target in dataloader:
        model.zero_grad()
        log_probs = model(context)
        loss = loss_function(log_probs, target)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if epoch % 50 == 0:
        print(f'Epoch {epoch}, Loss: {total_loss / len(dataloader)}')
```

SIMPLE ONE-LAYER LSTM

```
72  # Prediction function
73  def predict(text):
74      words = text.lower().replace(',', '').split()
75      input_idx = [vocab.get(word, 0) for word in words[-context_size:]]
76      input_tensor = torch.tensor([input_idx], dtype=torch.long)
77      with torch.no_grad():
78          log_probs = model(input_tensor)
79      return max(zip(log_probs[0].exp(), vocab.keys()), key=lambda p: p[0])[1]
80
81  # Test the prediction function
82  print(predict("teach me and"))  # Expected to output 'I', given the training context.
83
```

```
Epoch 0, Loss: 2.3626063466072083
Epoch 50, Loss: 1.2420078814029694
Epoch 100, Loss: 0.6485145222395658
Epoch 150, Loss: 0.38657374307513237
Epoch 200, Loss: 0.27861463837325573
Epoch 250, Loss: 0.2532136728987098
i
```

APPLICATIONS OF RNN (LSTM)

- Text generation
- Robot control
- Time series prediction
- Speech recognition
- Rhythm learning
- Music composition
- Grammar learning
- Handwriting recognition
- Human action recognition
- Sign language translation
- Protein homology detection
- …

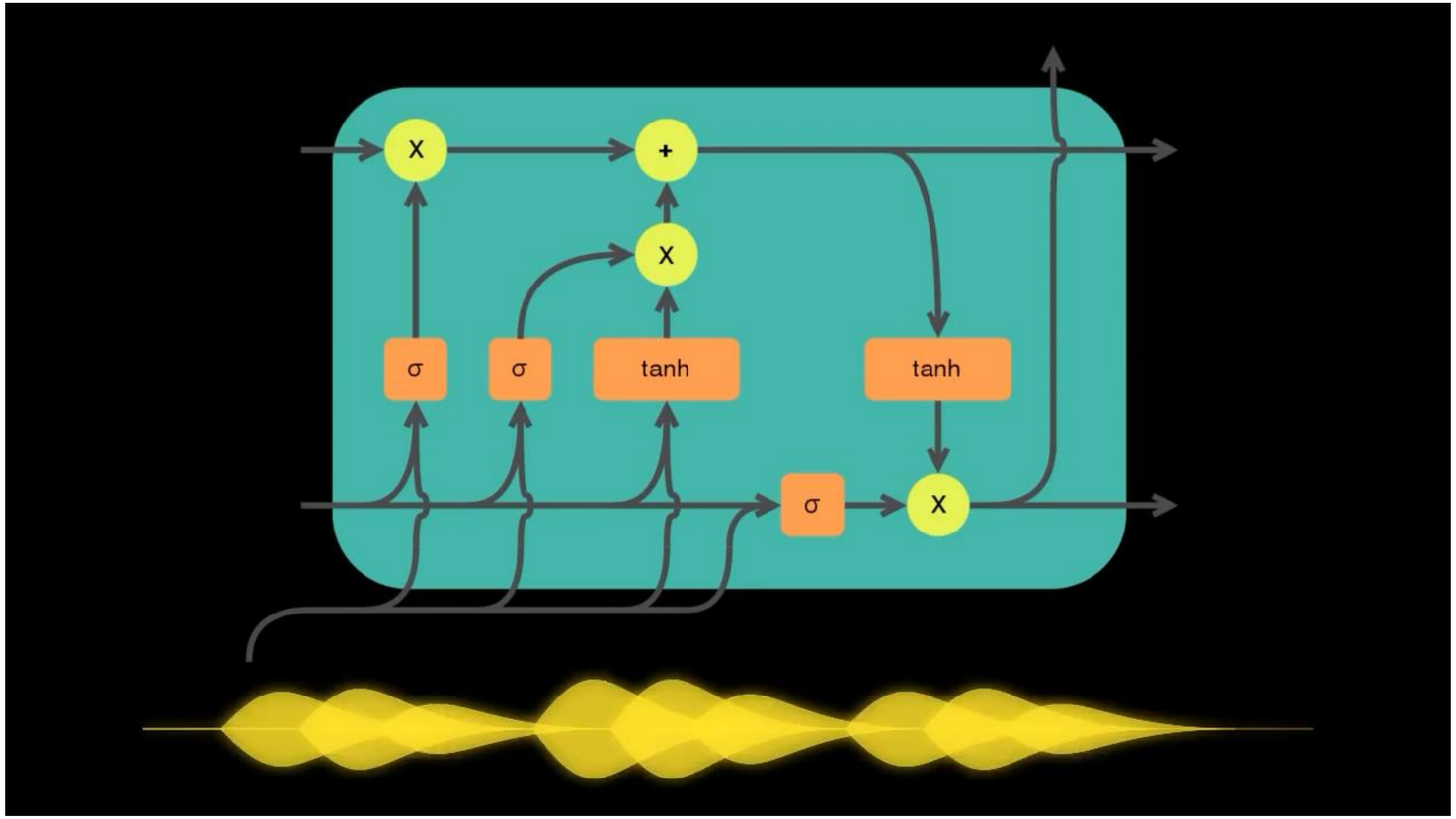
The first science fiction movie written by AI bot (LSTM)

Sunspring: an experimental science fiction short film entirely written by an artificial intelligence bot (called Benjamin) using neural network (using LSTM) – 2016

More: https://en.wikipedia.org/wiki/Sunspring

Music generated by LSTM

Source: https://github.com/chenfengw/AI-composer

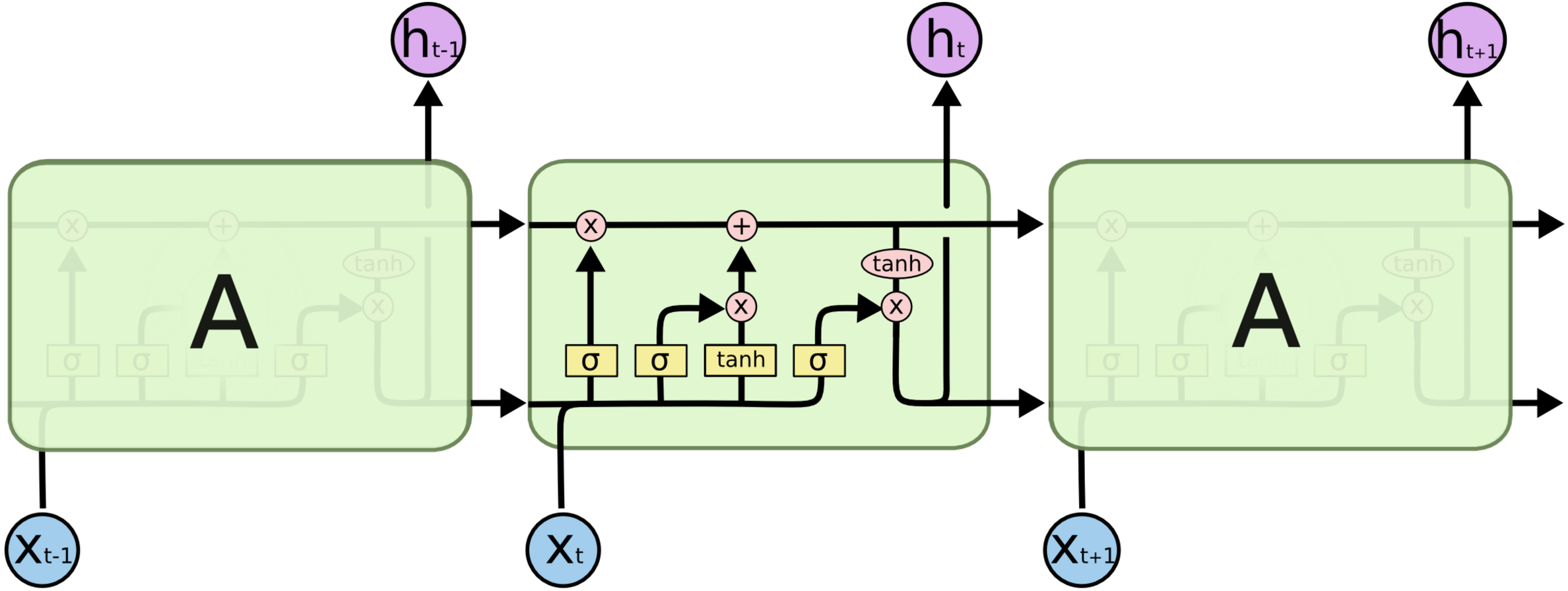


The Trailer for the World's First Fully AI-Generated Film is Here

APR 16, 2024 MATT GROWCOOT

https://petapixel.com/2024/04/16/the-trailer-for-the-worlds-first-fully-ai-generated-film-is-here/

Are there any problems with LSTM? What do you think?

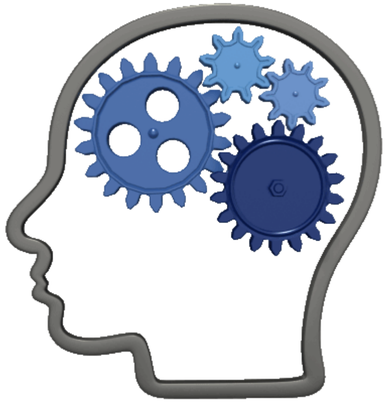


Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: RNN & LSTM

1. What is the main purpose of using RNNs?

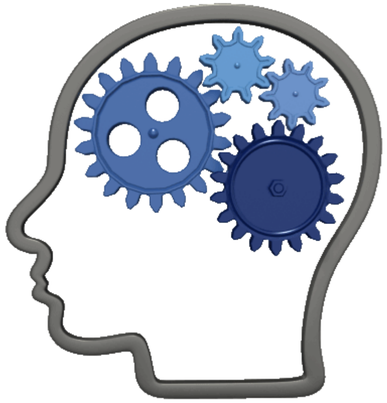
RNNs are designed to handle sequential data by maintaining a hidden state that allows them to capture temporal dependencies.



Recap: RNN & LSTM

2. What are the limitations of standard RNNs?

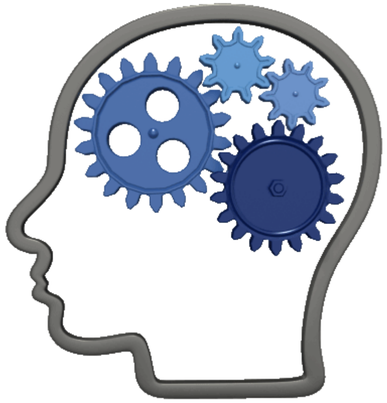
Standard RNNs suffer from the vanishing gradient problem, making it difficult to capture long-term dependencies in the data.



Recap: RNN & LSTM

3. How does LSTM overcome the vanishing gradient problem?

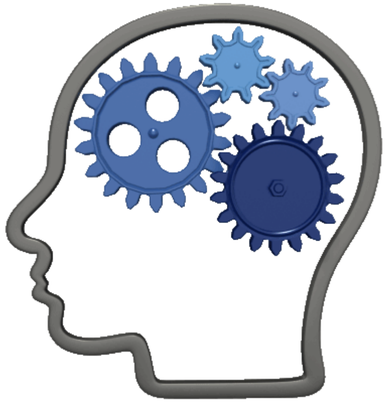
LSTM introduces a memory cell and three types of gates (input, forget, and output) to control the flow of information, allowing it to capture long-term dependencies effectively.



Recap: RNN & LSTM

4. What is the purpose of the input gate in an LSTM cell?

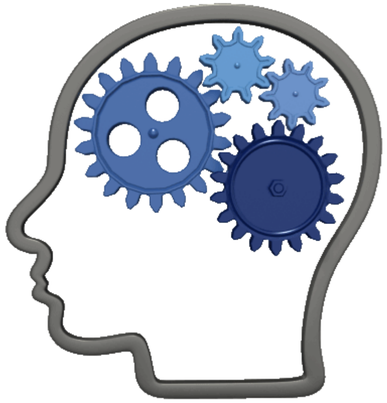
The input gate in an LSTM cell determines how much of the input information should be stored in the memory cell.



Recap: RNN & LSTM

5. Explain the forget gate in LSTM and its role.

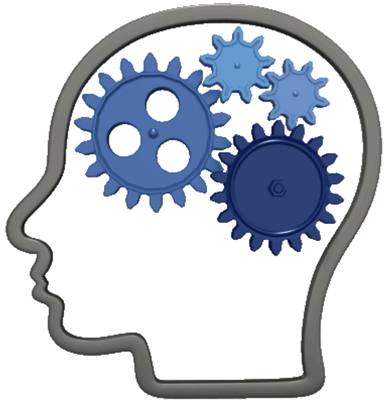
The forget gate controls the amount of information that should be discarded from the memory cell, enabling the model to forget irrelevant information.



Recap: RNN & LSTM

6. What are the advantages of using LSTM over standard RNNs?

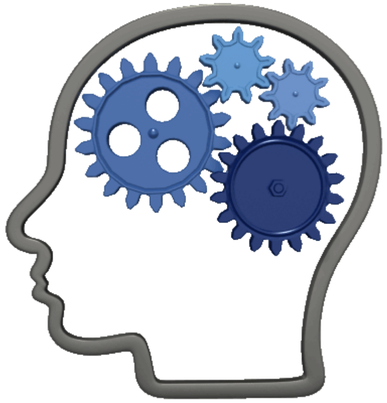
LSTM can capture long-term dependencies more effectively due to its gating mechanisms, making it suitable for tasks involving long sequences of data.



Recap: RNN & LSTM

7. Can LSTMs handle variable-length sequences?

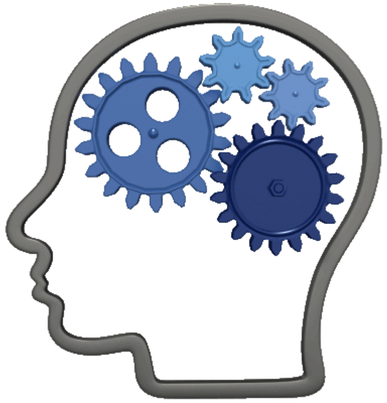
Yes, LSTMs are designed to handle variable-length sequences by allowing them to process inputs of different lengths and maintain context information through the cell state.



Recap: RNN & LSTM

8. What is the major drawback of using LSTM or RNN architectures?

LSTM and RNN architectures can be computationally expensive and suffer from longer training times compared to other models.



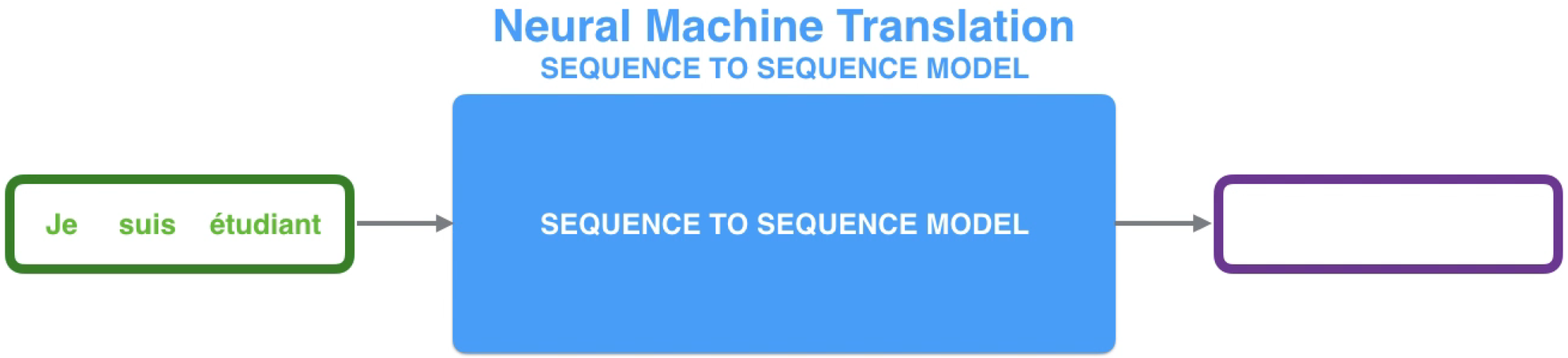
LEARN MORE: ANN & RNN

ANN: https://www.youtube.com/watch?v=MfIjxPh6Pys
RNN: https://www.youtube.com/watch?v=6niqTuYFZLQ

Sequence-to-Sequence Models

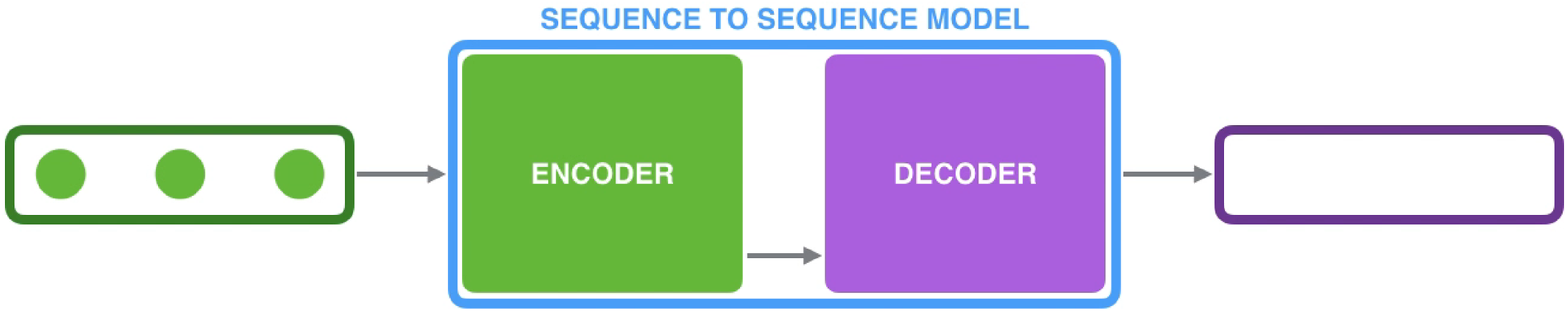
Sequence-to-Sequence (Seq2Seq) models are a class of models that can transform an input sequence into an output sequence, such as translating a sentence from one language to another.

Application examples: Machine translation, text summarization, speech recognition, etc.



Seq2Seq Model Overview

Architecture: The Seq2Seq model consists of two main components: an encoder and a decoder.
Encoder: Processes the input sequence and produces a fixed-length representation (context vector) that captures the input's meaning.
Decoder: Takes the context vector and generates the output sequence, one element at a time.



Encoder in Seq2Seq Model

Input Embedding: Converts input sequence elements into dense vectors.
Encoder RNN: Processes the embedded input sequence, maintaining hidden states at each step.
Final Hidden State: Represents the context vector that summarizes the input sequence.

```python
import torch
import torch.nn as nn

# Define an example encoder
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output, hidden = self.gru(embedded, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size)
```

Decoder in Seq2Seq Model

Output Embedding: Converts output sequence elements into dense vectors.
Decoder RNN: Processes the embedded output sequence, maintaining hidden states at each step.
Output Generation: Produces the output sequence element by element, conditioning on the context vector and previously generated elements.

```python
# Define an example decoder
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden
```

Limitations of Seq2Seq Models

- **Fixed-Length Context Vector:** The Seq2Seq model's fixed-length context vector may struggle with capturing long-range dependencies in the input sequence.
- **Information Compression:** The encoder compresses the entire input sequence into a fixed-length vector, potentially losing important details.
- **Lack of Alignment:** Seq2Seq models don't explicitly learn the alignment between input and output sequences, which can be problematic for translation tasks.

Introduction to Self-Attention

- **Motivation:** Self-attention mechanisms aim to address the limitations of Seq2Seq models by capturing dependencies between different positions in the input sequence.
- **Key Idea:** Assigns weights to different positions in the input sequence to compute a weighted sum, allowing the model to focus on relevant information.

THE SENTENCE SYNTHESIS CHALLENGE

(Source: google.com)

THE SENTENCE SYNTHESIS CHALLENGE

Objective: Use teamwork and individual analysis to reconstruct a sentence from scattered words, mimicking the processes in a Transformer model.

Game Rules and Steps:

- Each student receives one index card with a word on it. This card is your starting point.

Phase 1: Multi-Head Attention Phase (Initial Data Collection)

- **Duration:** 5 minutes.
- **Goal:** Mingle with classmates to collect as many words as possible. Write down each word you gather. At this stage, focus on collecting words without worrying about their order.

Phase 2: Individual Processing Phase 1 (Reflection and Hypothesis)

- **Duration:** 3 minutes.
- **Goal:** Think about possible positions for your collected words within a sentence. Use your knowledge of grammar and sentence structure to form a preliminary sequence.

THE SENTENCE SYNTHESIS CHALLENGE

Phase 3: Multi-Head Attention Phase 2 (Enhanced Data Collection)

- **Duration:** 5 minutes.
- **Goal:** With your initial sentence ideas in mind, seek out specific words you need to complete or correct your sentence. Check and adjust your sequence based on the new information you gather.

Phase 4: Individual Processing Phase 1 (Reflection and Hypothesis)

- **Duration:** 3 minutes.
- **Goal:** Think about possible positions for your collected words within a sentence. Use your knowledge of grammar and sentence structure to form a preliminary sequence.

Continuous learning fuels personal growth, enhances professional skills, fosters innovation, broadens perspectives, strengthens adaptability, and ultimately empowers individuals to navigate the complexities of the modern, ever-evolving global landscape effectively.

Discussion