MLAI 504
NEURAL NETWORKS & DEEP LEARNING

Dr. Zein Al Abidin IBRAHIM

zein.ibrahim@ul.edu.lb

RECURRENT NEURAL NETWORK (RNN)

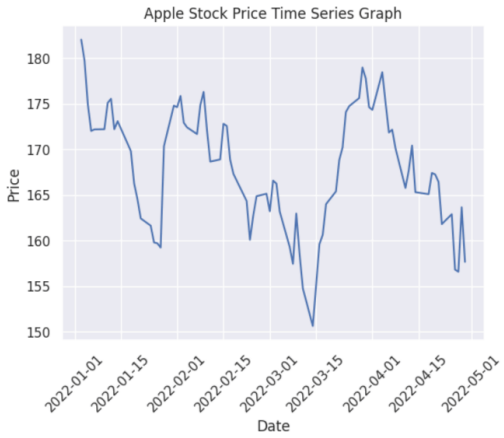
DEEP LEARNING

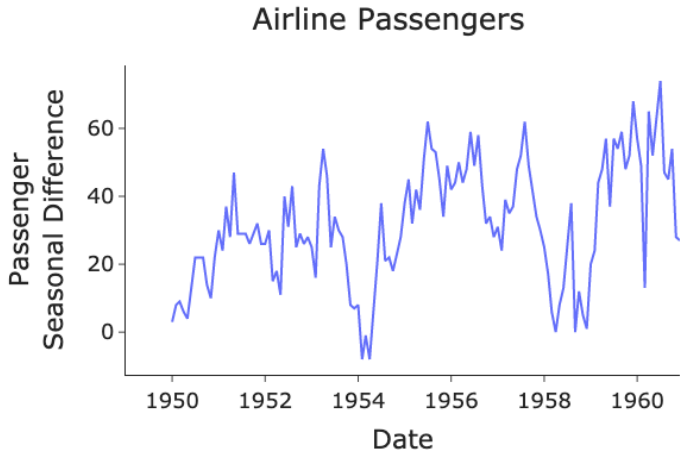
MAIN TYPES OF NEURAL NETWORKS

- **Feed-Forward Neural Network (FNN)**: Used for general Regression and Classification problems.
- **Convolutional Neural Network (CNN)**: Used for object detection and image classification and many computer vision applications.
- **Recurrent Neural Network (RNN)**: Used for speech recognition, voice recognition, time series prediction, and natural language processing.
- **Generative Adversarial Network (GAN)**: Used for tasks like image generation, text-to-image synthesis, and style transfer.
- ….

UNDERSTANDING SEQUENTIAL DATA

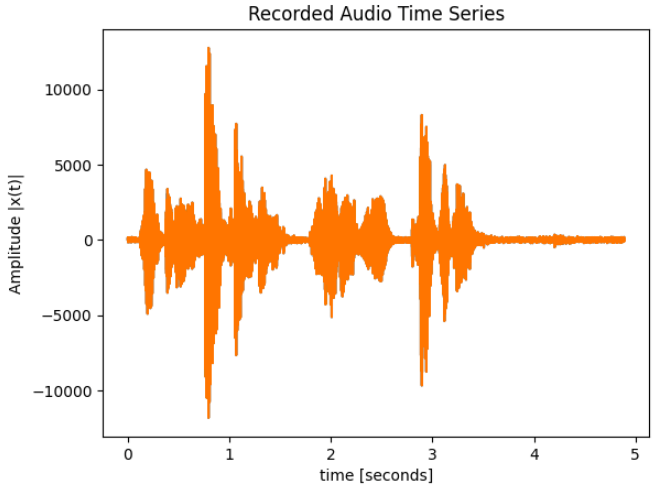
- **What is Sequential Data?**
  - Data points arranged in a specific order, where the sequence itself carries meaning.
  - The relationship between elements is crucial for understanding the data.
- **Examples:**
  - **Time Series Data:**
    - Stock prices over time (Chart showing stock price fluctuations over time)
    - Sensor readings from a machine (Illustration of sensor data readings plotted chronologically)
    - Daily weather observations (Image of weather data graphed over weeks or months)

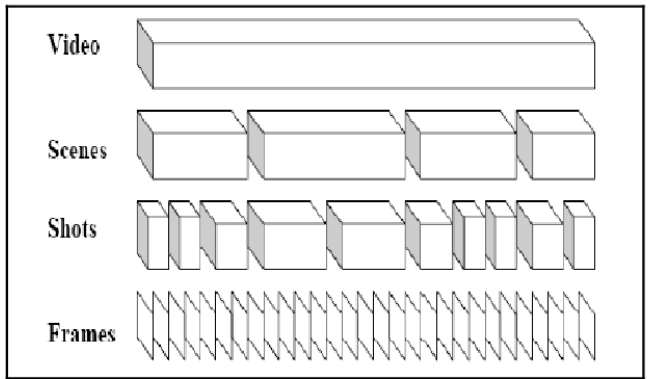




UNDERSTANDING SEQUENTIAL DATA

- **Textual Data:**
  - Sentences in a book or article
  - Conversations in a chat log
  - Genetic sequences (DNA or RNA strands)
- **Audio Data:**
  - Speech recordings
  - Music pieces
- **Video Data:**
  - Sequences of frames in a video
  - Gesture recognition data (Video clip of a person performing gestures)





WHAT ARE RNNS

- **Recurrent Neural Networks (RNNs)**
  - type of ANN designed to process sequences of data.
  - Work especially well for jobs requiring sequences, such as time series data, voice, natural language, and other activities.
- **Idea**
  - Works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer for the next input.

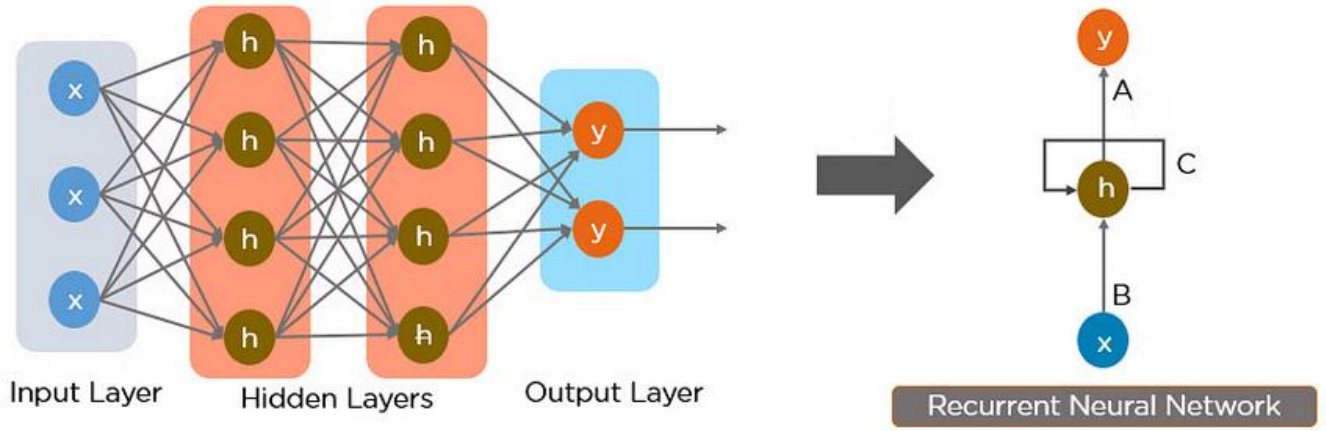
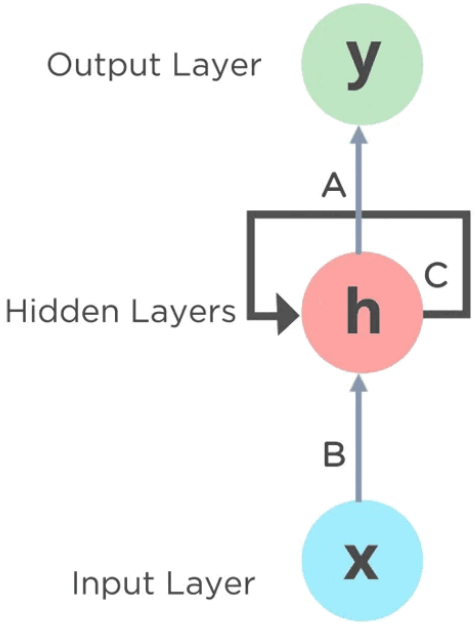


Fig: Simple Recurrent Neural Network

WHAT ARE RNNS

- **Recurrent Neural Networks (RNNs)**
  - Nodes in different layers are compressed to form a single layer of recurrent neural networks.
  - Called the Vanilla RNNs which are **the simplest form of RNN.**
  - A, B, and C are the parameters of the network.



A, B and C are the parameters

WHAT ARE RNNS

- **Recurrent Neural Networks (RNNs)**
  - "x" is the input layer, "h" is the hidden layer, and "y" is the output layer.
  - A, B, and C are the network parameters used to improve the output of the model.
  - At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.

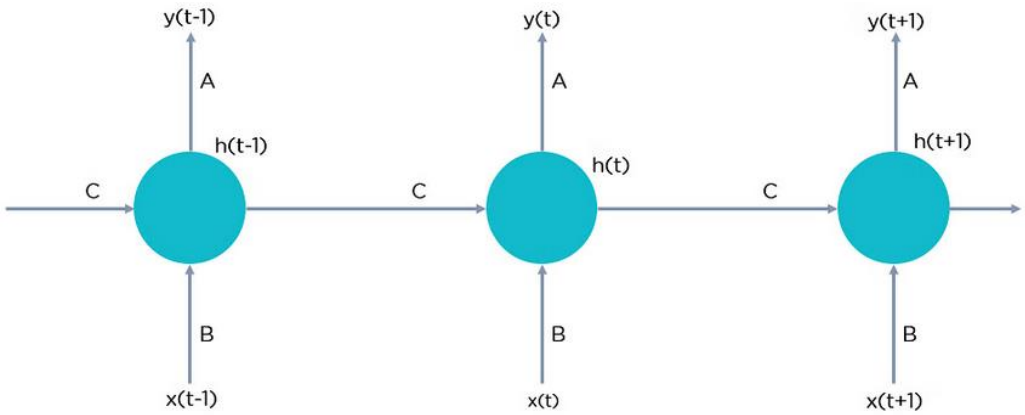
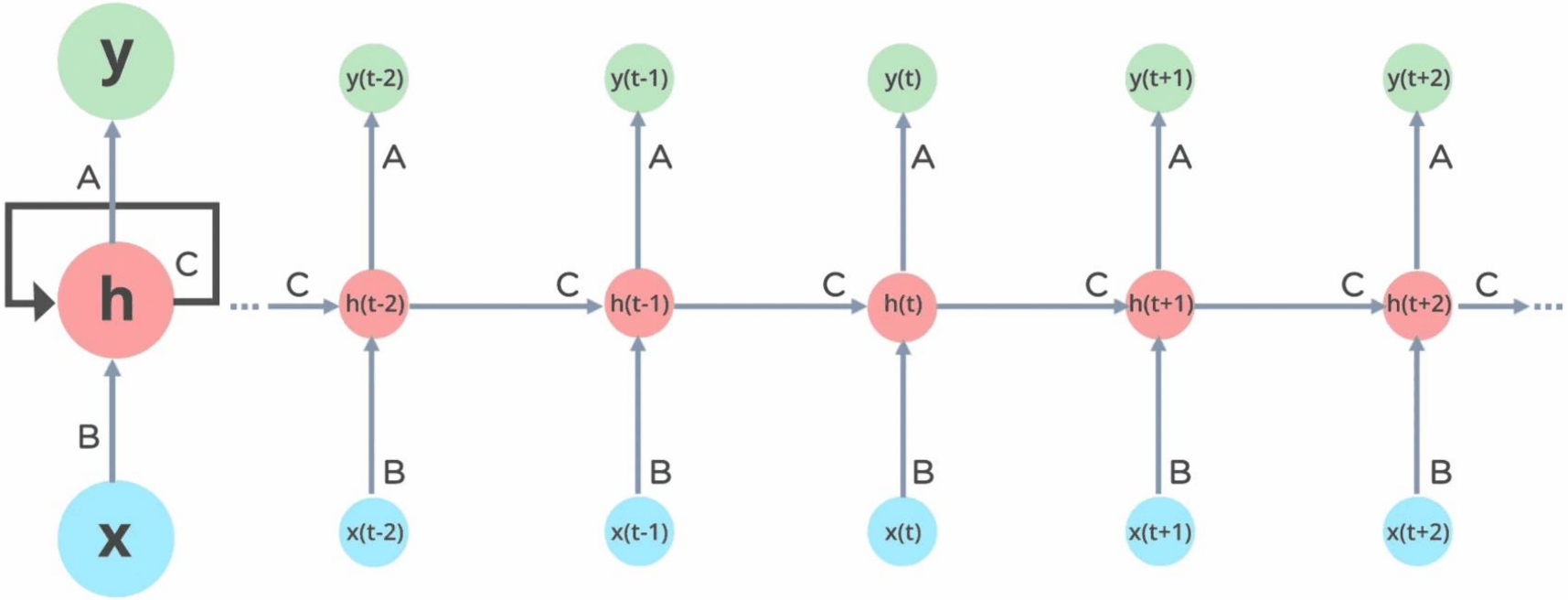
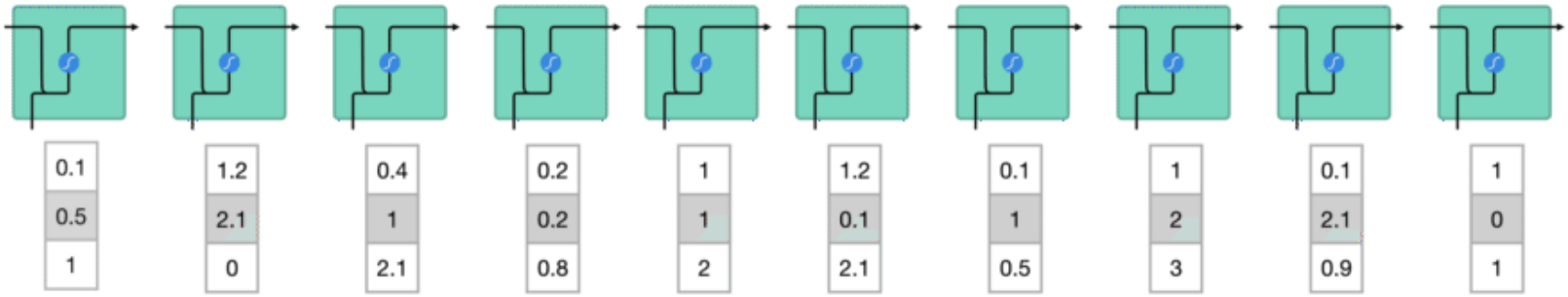


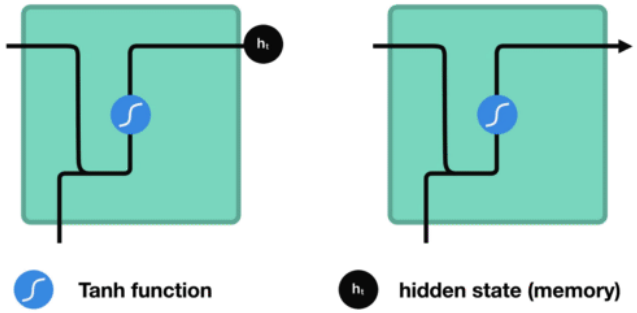
Fig: Fully connected Recurrent Neural Network

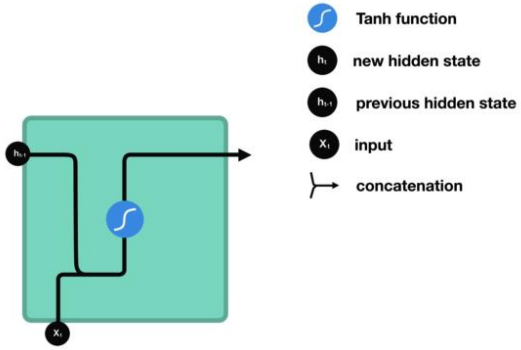
WHY RECURRENT NEURAL NETWORKS ?

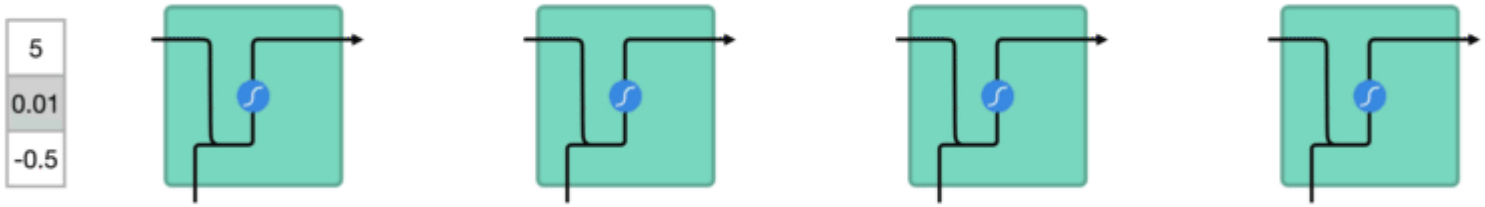
- **Existing networks such as FNN and CNN**
  - Cannot handle sequential data
  - Considers only the current input
  - Cannot memorize previous inputs
- How RNN works ?




0.1
0.5
1
1.2
2.1
0
0.4
1
2.1
0.2
0.2
0.8
1
1
2
1.2
0.1
2.1
0.1
1
0.5
1
2
3
0.1
2.1
0.9
1
0
1


Tanh function
hidden state (memory)


Tanh function
new hidden state
previous hidden state
input
concatenation


5
0.01
-0.5

POSSIBLE APPLICATIONS

- Image Captioning

"A Dog catching a ball in mid air"

POSSIBLE APPLICATIONS

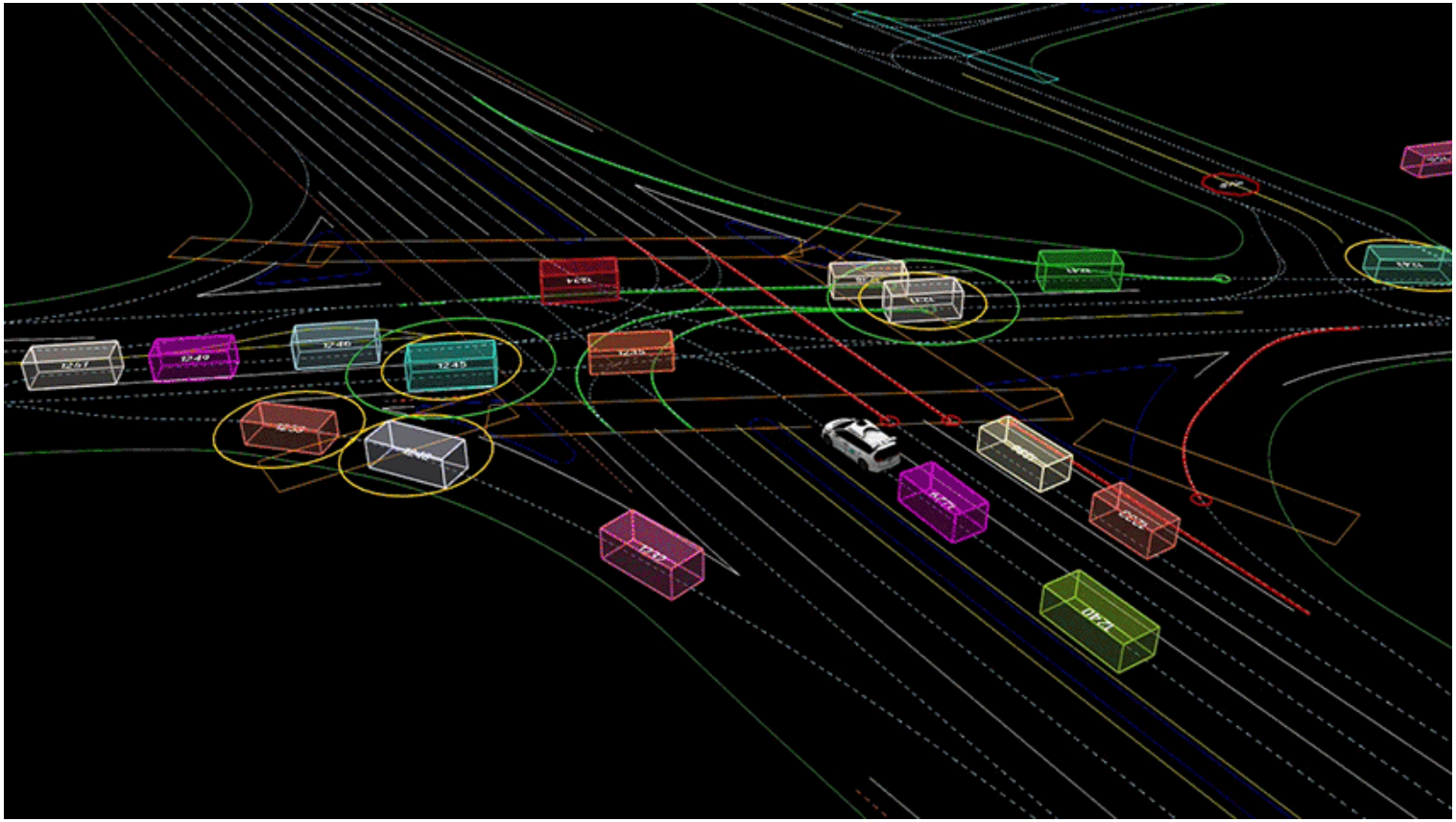
- Speech to text

POSSIBLE APPLICATIONS

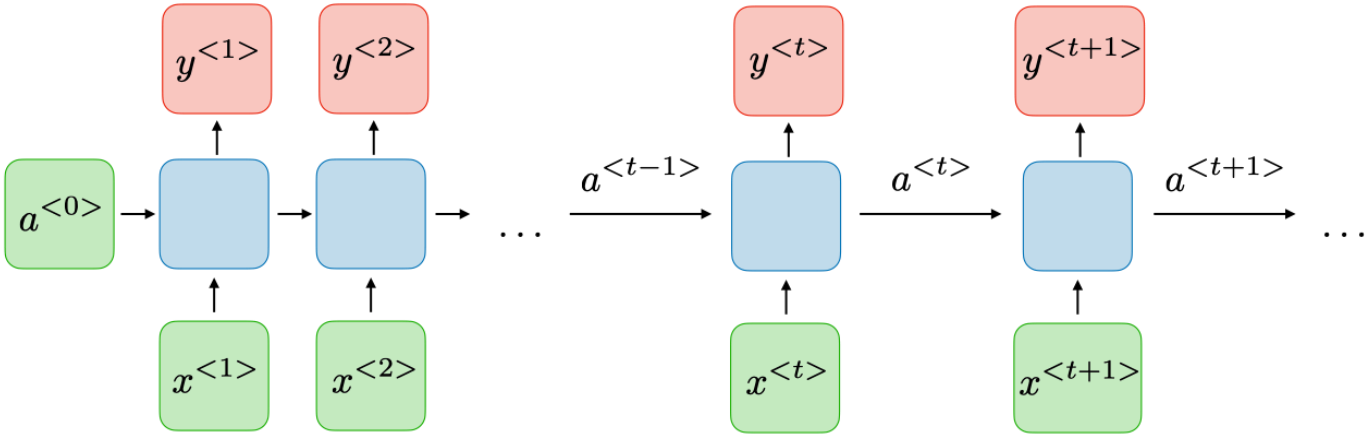
- Machine translation

POSSIBLE APPLICATIONS

- Video understanding like objects tracking



MORE ABOUT RNN

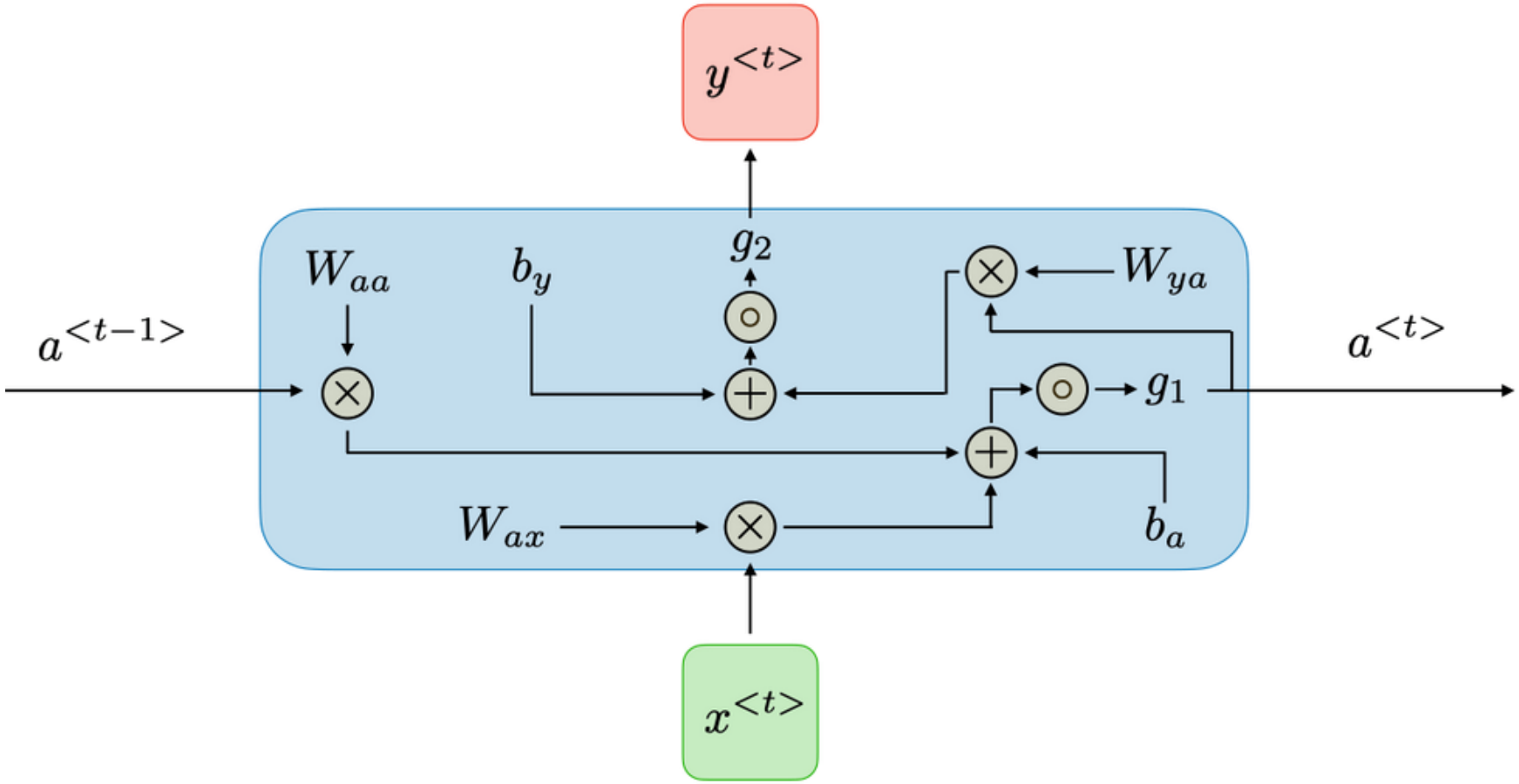


- For each timestamp t, the activate $a^{<t>}$ and the output $y^{<t>}$ are calculated as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

MORE ABOUT RNN



SOME ADVANTAGES OF RNN

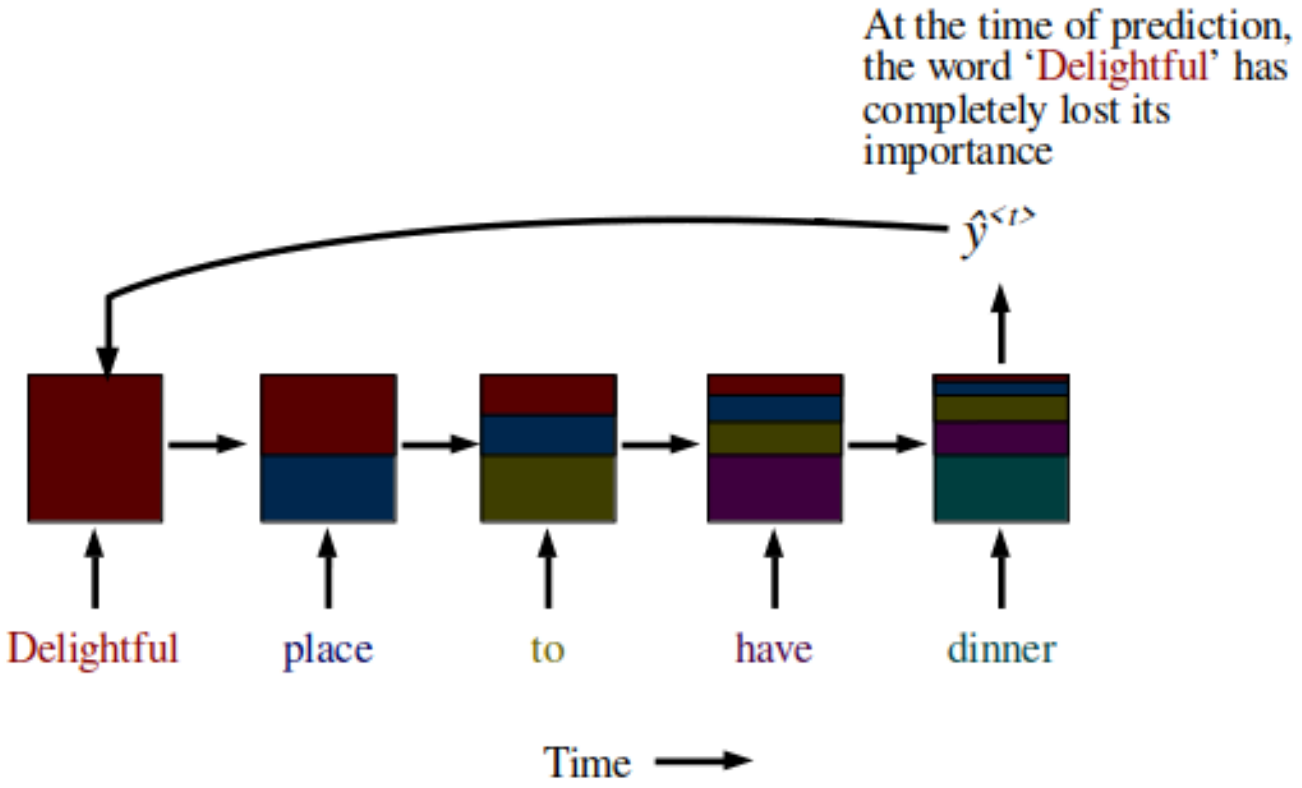
- **Sequential Data Processing:**
  - RNNs are designed to handle sequential data, making them well-suited for tasks where the order of input elements matters.
- **Variable-Length Input Sequences:**
  - RNNs can handle input sequences of variable lengths,
- **Parameter Sharing:**
  - RNNs use the same set of weights across different time steps, allowing them to share parameters and capture patterns that occur at different positions in the sequence.
- **Memory of Previous Inputs:**
  - RNNs have a form of memory that allows them to retain information from previous time steps.
- **Online Learning:**
  - RNNs can be trained online or incrementally, which means they can update their parameters as new data becomes available.

SOME DISADVANTAGES OF RNN

- **Vanishing and Exploding Gradients:**
  - Traditional RNNs can face issues with vanishing gradients or exploding gradients during training. This makes it difficult for the network to learn long-term dependencies or effectively update weights.
- **Difficulty with Long-Term Dependencies:**
  - RNNs struggle to capture long-term dependencies in sequential data.
- **Difficulty in Capturing Context:**
  - RNNs may have difficulty capturing and maintaining contextual information over long sequences,
- **Sensitivity to Input Order:**
  - The order of inputs in the sequence is critical for RNNs. If the order is changed, the network may produce different outputs.
- **Lack of Parallelism:**
  - RNNs inherently process sequences one time step at a time, limiting their ability to take advantage of parallel processing.

DISADVANTAGE: EXAMPLE

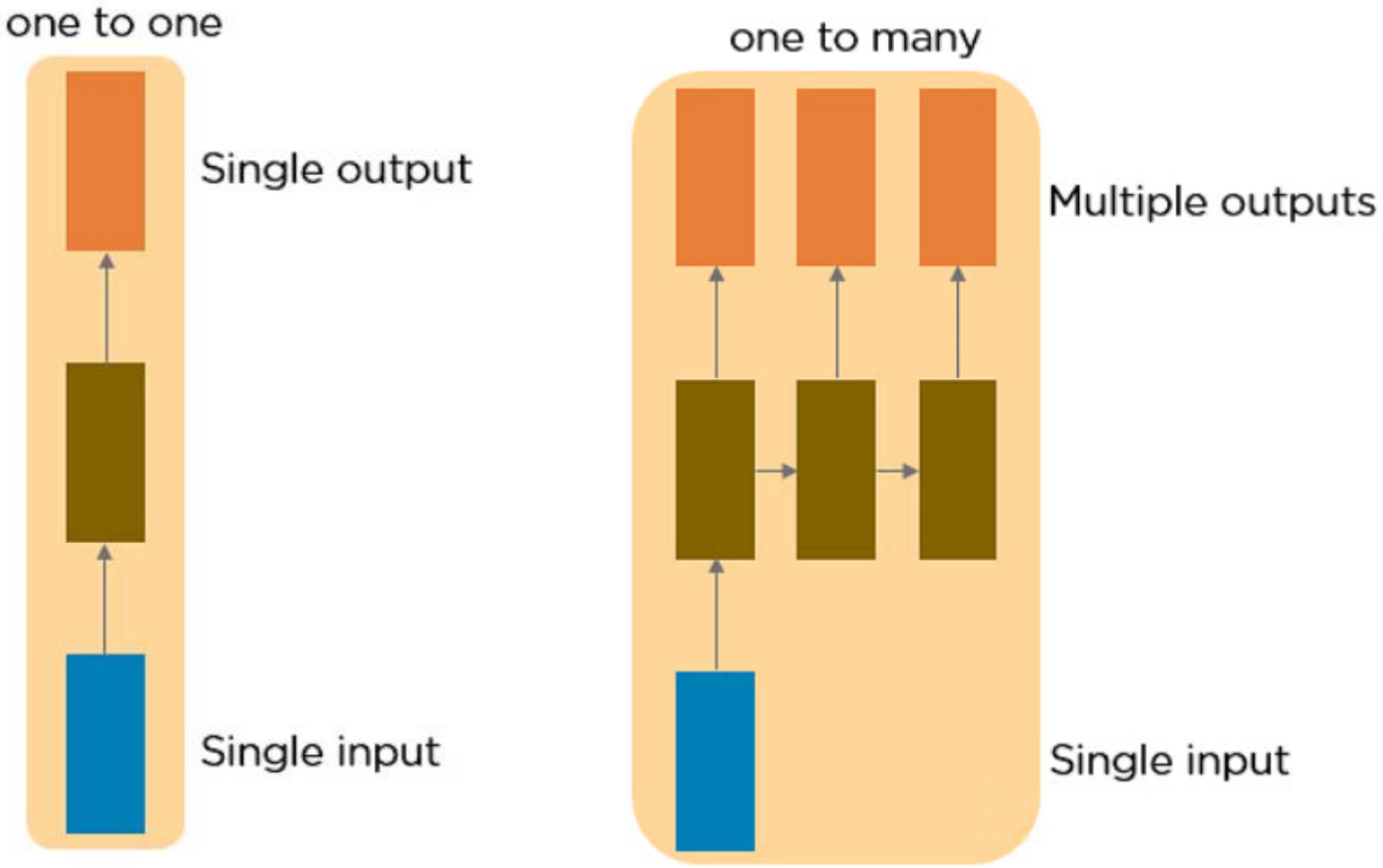
Vanilla RNNs can learn **short-term dependencies**, but they struggle with **long-term dependencies.**



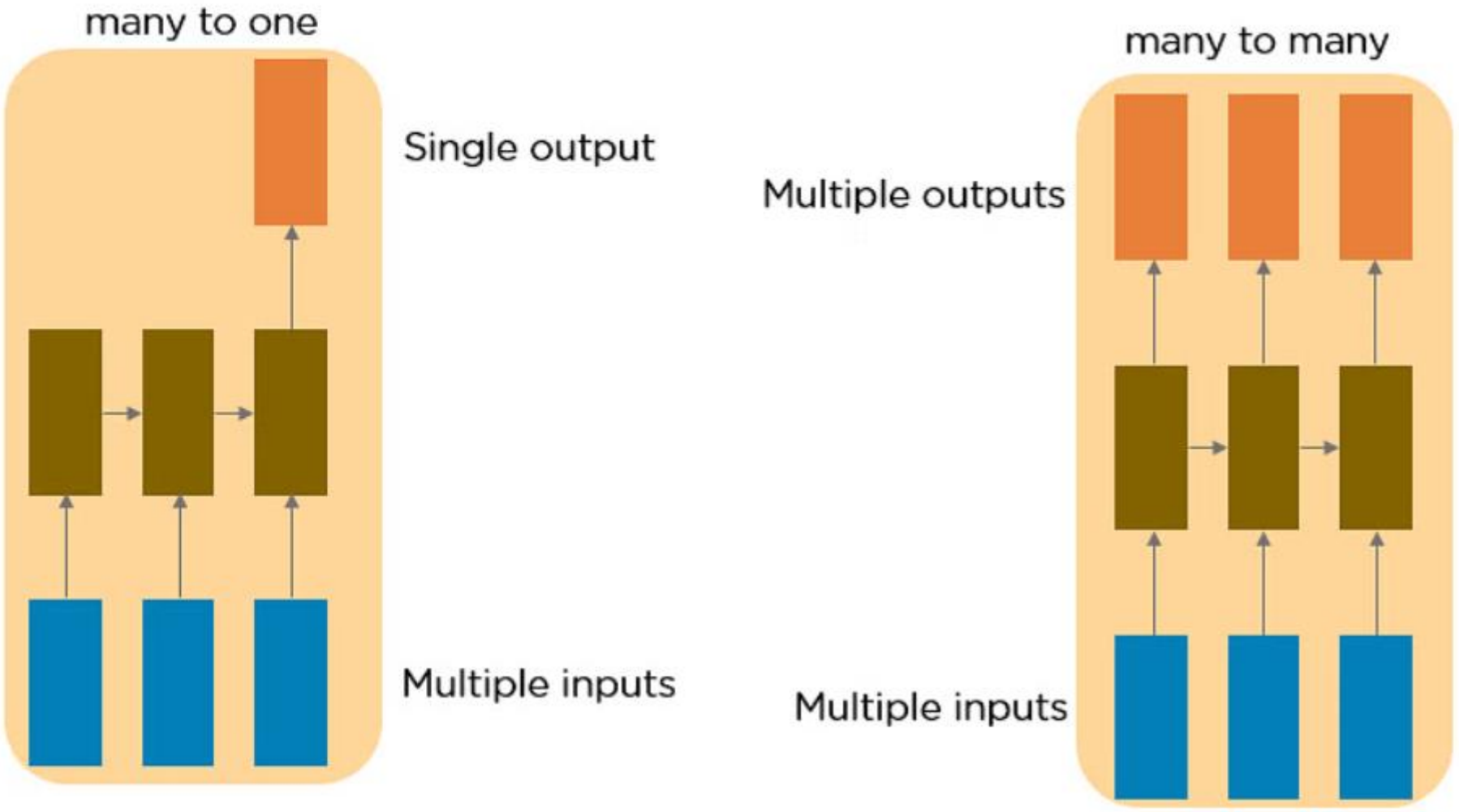
TYPES OF RECURRENT NEURAL NETWORKS

- One to One
  - Ex: Image classification (one image → one class)
- One to Many
  - Ex: Image captioning (one image → sequence of words)
- Many to One
  - Ex: Sentiment analysis (sequence of words → sentiment)
- Many to Many
  - Ex: Named entity recognition (sequence of words → sequence of entities)

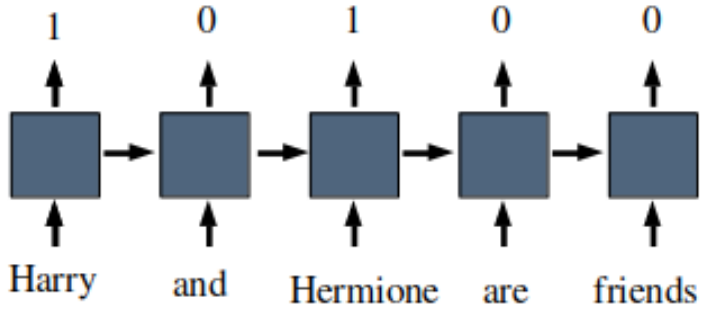
TYPES OF RECURRENT NEURAL NETWORKS



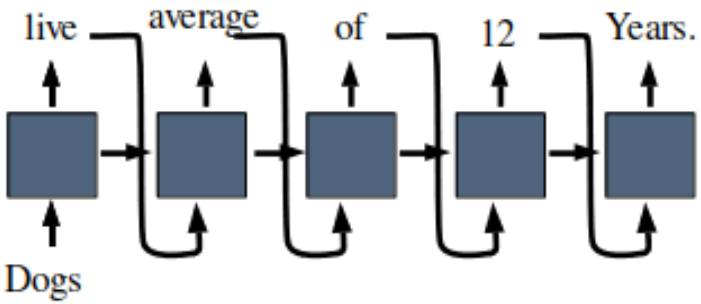
TYPES OF RECURRENT NEURAL NETWORKS



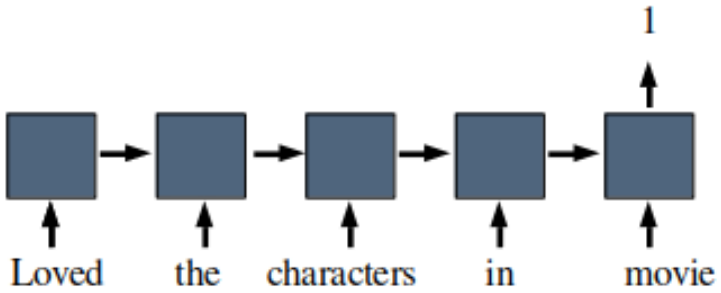
EXAMPLES



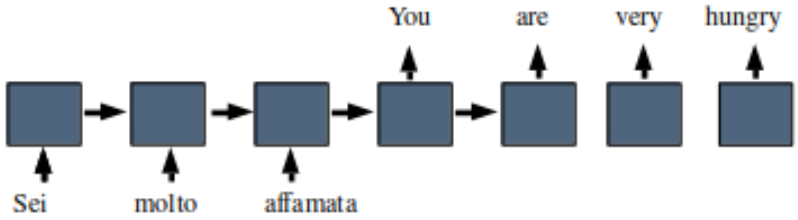
Many to many (equal input-output)



One to many

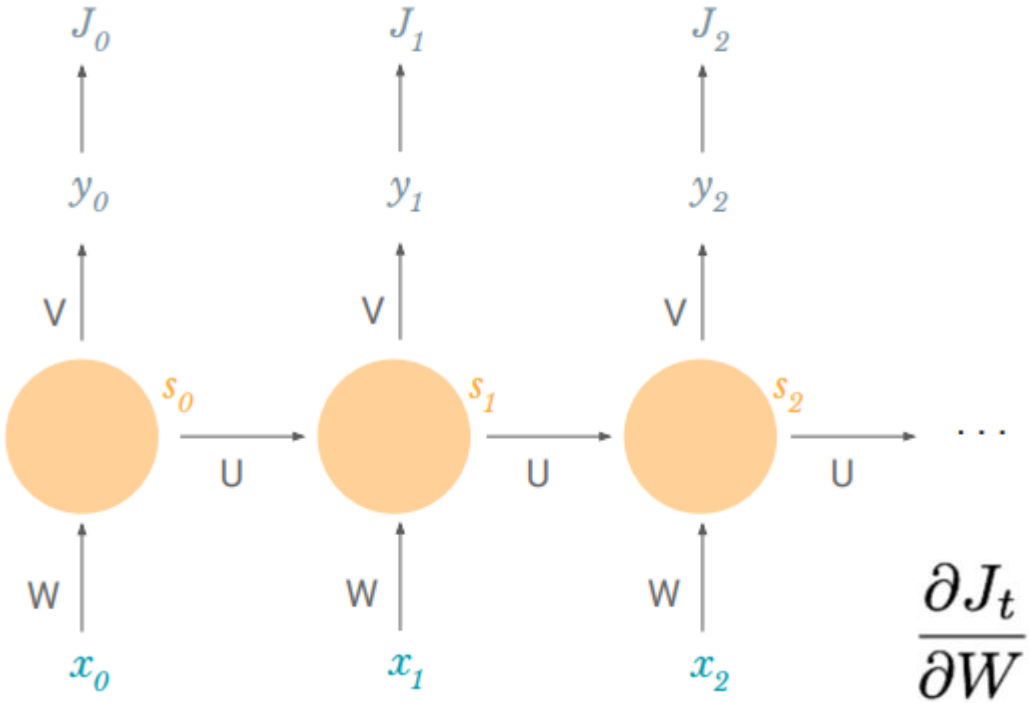


Many to one



Many to many (unequal input-output)

TRAINING USING BACKPROPAGATION THROUGH TIME (BPTT)



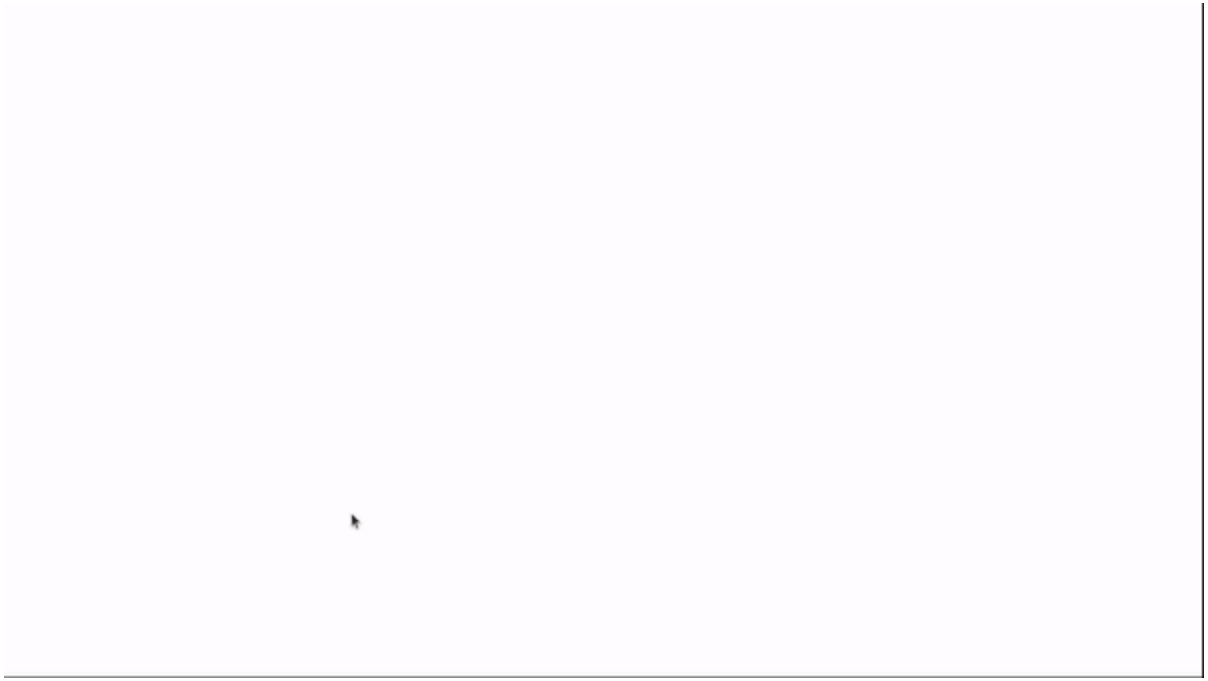
$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

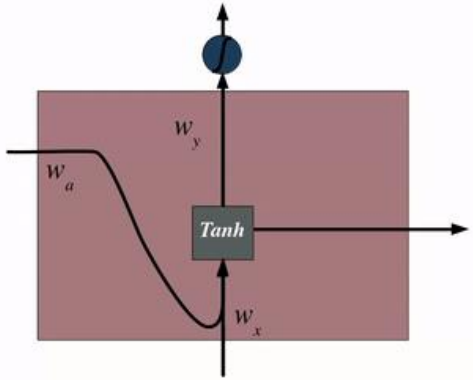
$$\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t}\frac{\partial y_t}{\partial s_t}\frac{\partial s_t}{\partial s_k}\frac{\partial s_k}{\partial W}$$

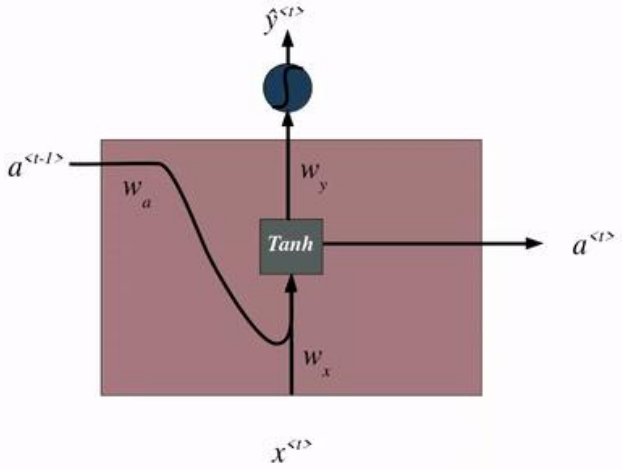
loss at time t = $J_t(\Theta)$ total loss = $J(\Theta) = \Sigma_t J_t(\Theta)$

EXAMPLE: TEXT

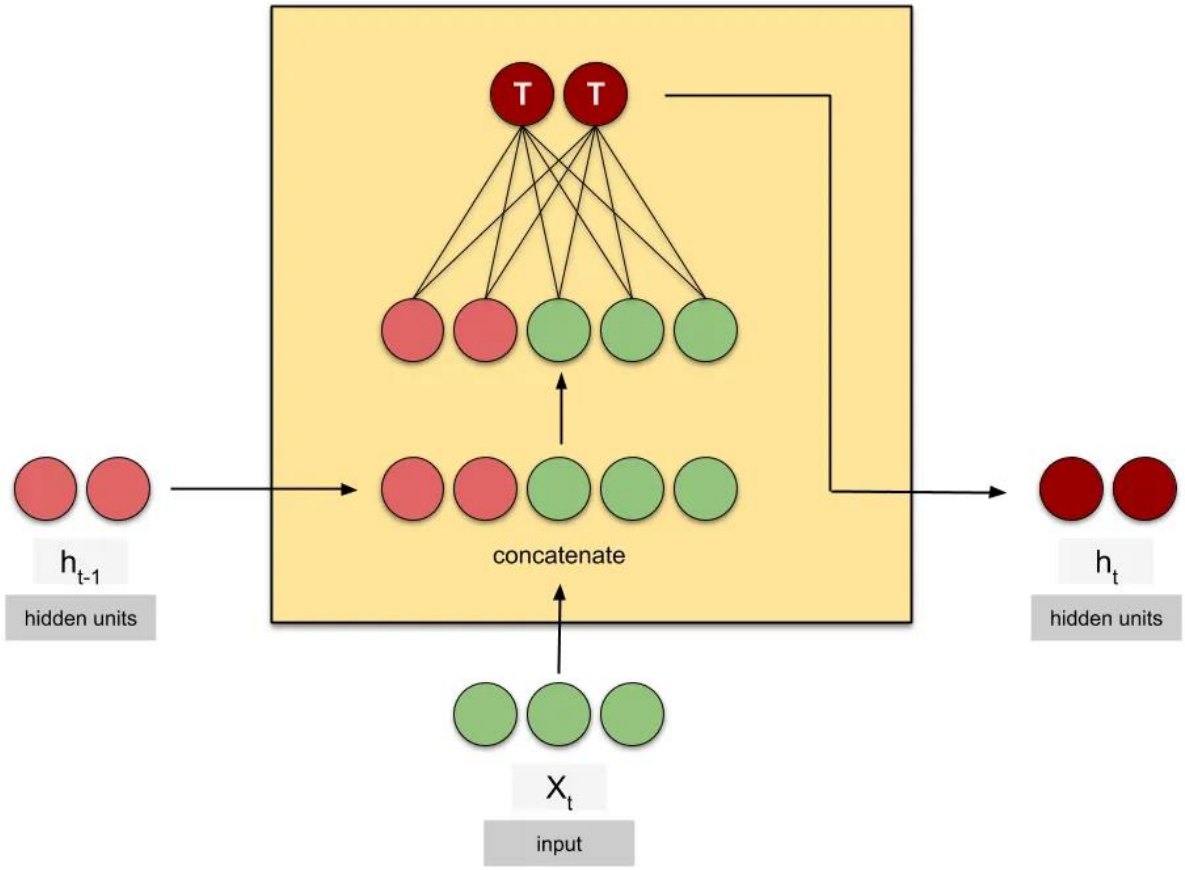


EXAMPLE





ILLUSTRATED FIGURE OF RNN



VARIANTS OF RNN

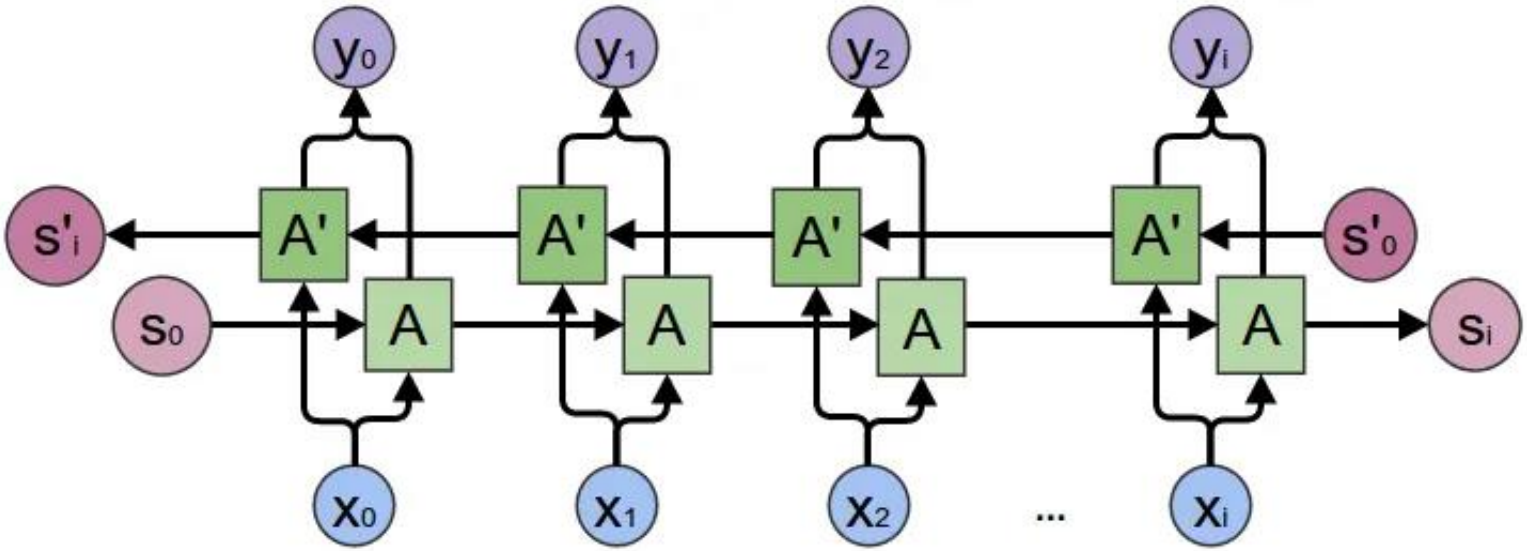
- To address the limitations of the standard RNN architecture → several variants
  1. **Long Short-Term Memory (LSTM) Networks**
     - Type of RNN that is designed to handle the vanishing gradient problem
     - Three gating mechanisms that control the flow of information through the network is used: the input gate, the forget gate, and the output gate.
     - These gates allow the LSTM network to selectively remember or forget information from the input sequence, which makes it more effective for long-term dependencies.
  2. **Gated Recurrent Unit (GRU) Networks**
     - Type of RNN that is designed to address the vanishing gradient problem.
     - It has two gates: the reset gate and the update gate.
     - The reset gate determines how much of the previous state should be forgotten, while the update gate determines how much of the new state should be remembered.

VARIANTS OF RNN

3. **Bidirectional RNNs:**
   - Designed to process input sequences in both forward and backward directions.
   - This allows the network to capture both past and future context.
4. **Encoder-Decoder RNNs:**
   - Consist of two RNNs encoder and decoder.
   - Encoder network that processes the input sequence and produces a fixed-length vector representation of the input.
   - Decoder network that generates the output sequence based on the encoder's representation.
   - This architecture is commonly used for sequence-to-sequence tasks such as machine translation.
5. **Attention Mechanisms**
   - A technique that can be used to improve the performance of RNNs on tasks that involve long input sequences.
   - Work by allowing the network to attend to different parts of the input sequence selectively rather than treating all parts of the input sequence equally.
   - Help the network focus on the input sequence's most relevant parts and ignore irrelevant information.

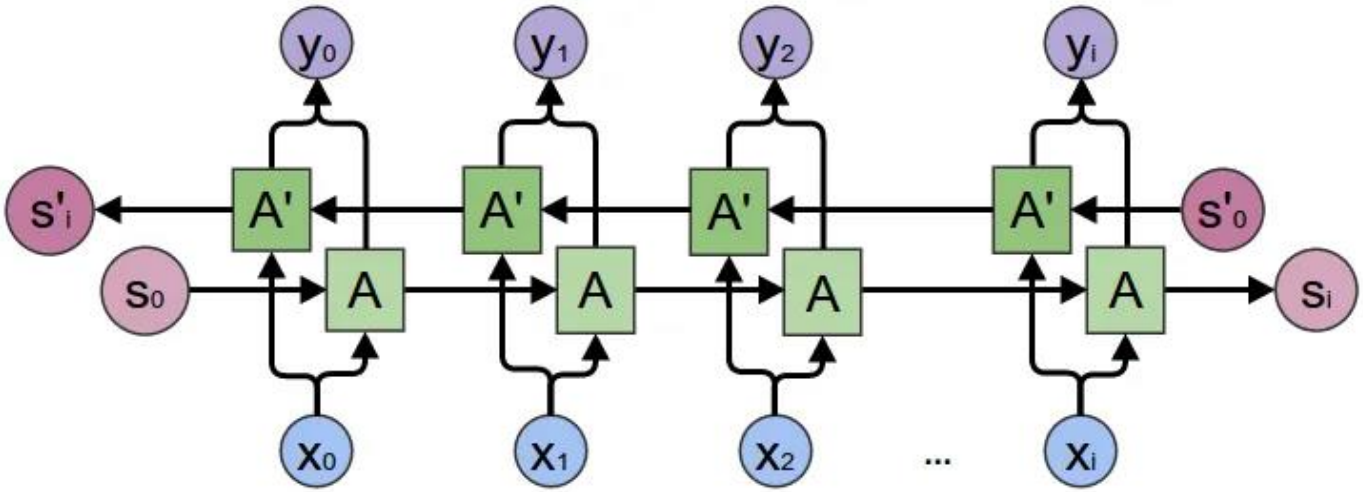
BIDIRECTIONAL RNN

- **Is RNN sufficient in all cases?**
  - ✓ I am ___.
  - ✓ I am ___ hungry.
  - ✓ I am ___ hungry, and I can eat half a cow.



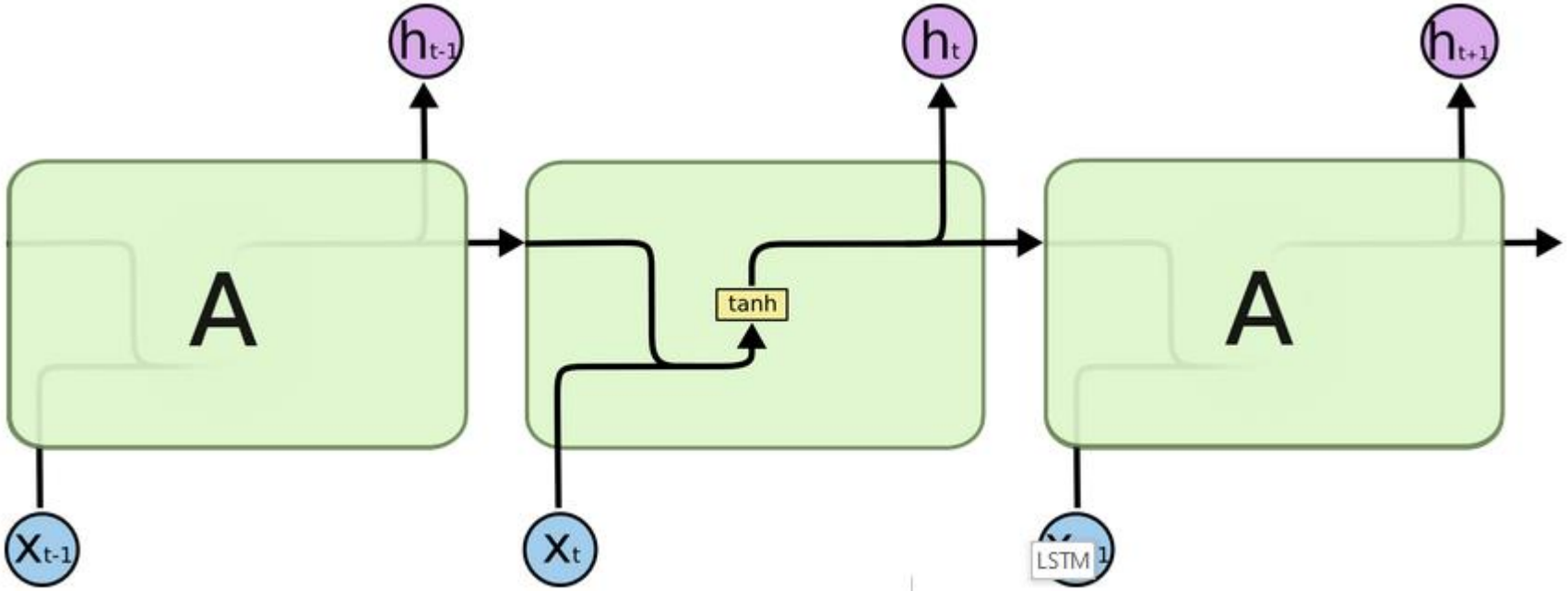
BIDIRECTIONAL RNN

- **Bidirectional RNN**
  - a combination of 2 individual unidirectional RNN.
  - One process the sequence from left to right and the other in the inverse order
  - The output (y_i) combination operators are:
    - Concatenation
    - Sum
    - Average
    - Maximum



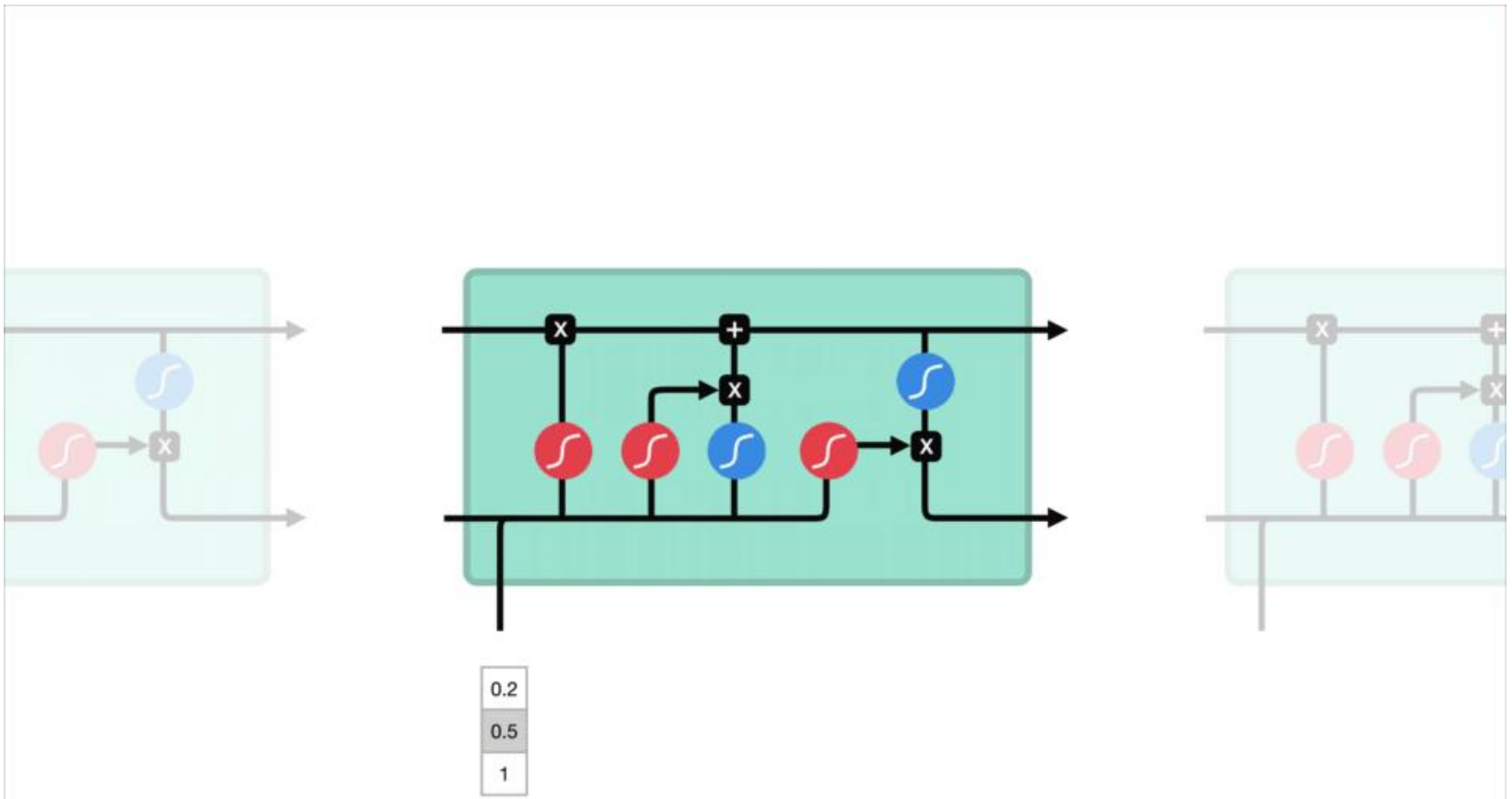
LSTM

- Traditional RNN → simple structure of a single layer



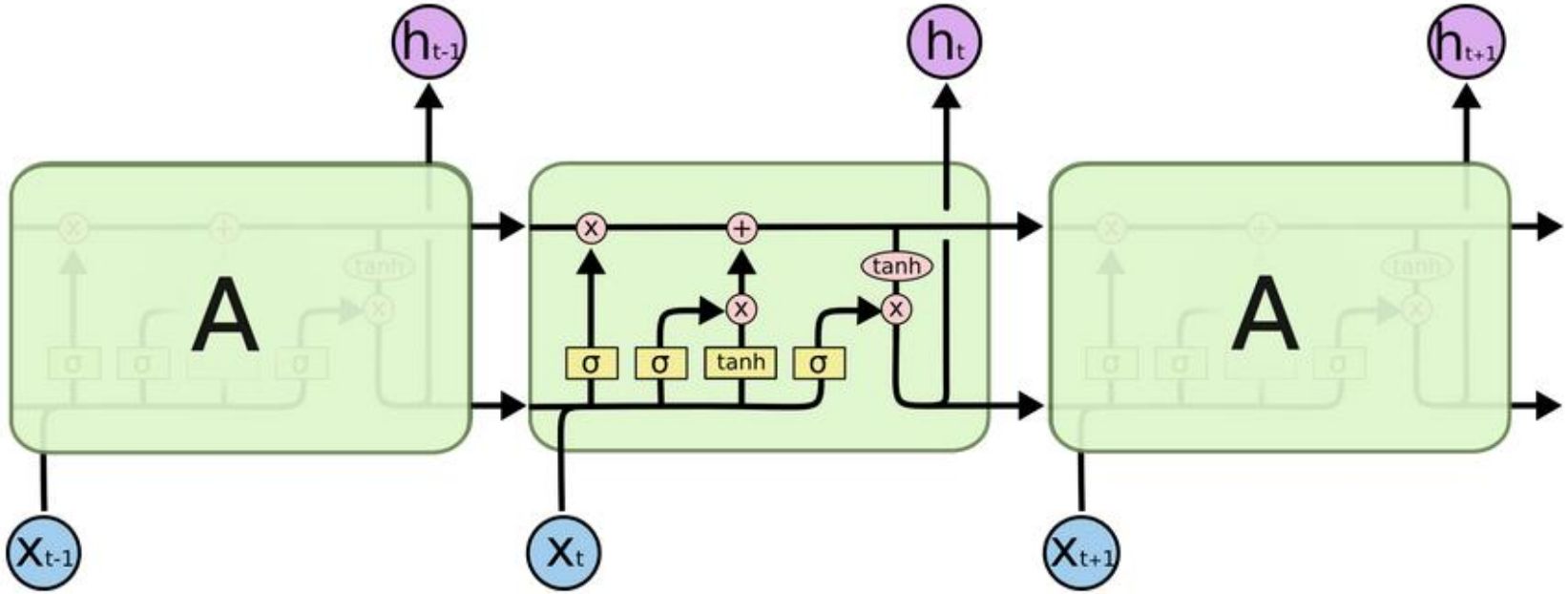
The repeating module in a standard RNN contains a single layer.

LSTM ANIMATED

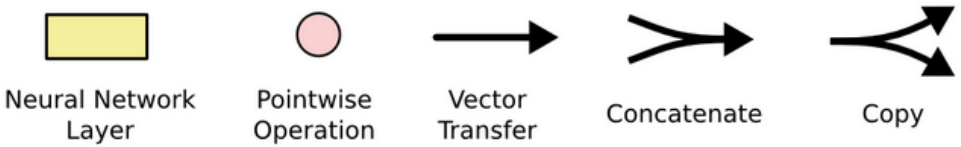


LSTM

- LSTM → repeating module is bit different
- In LSTM module → four interacting layers are communicating



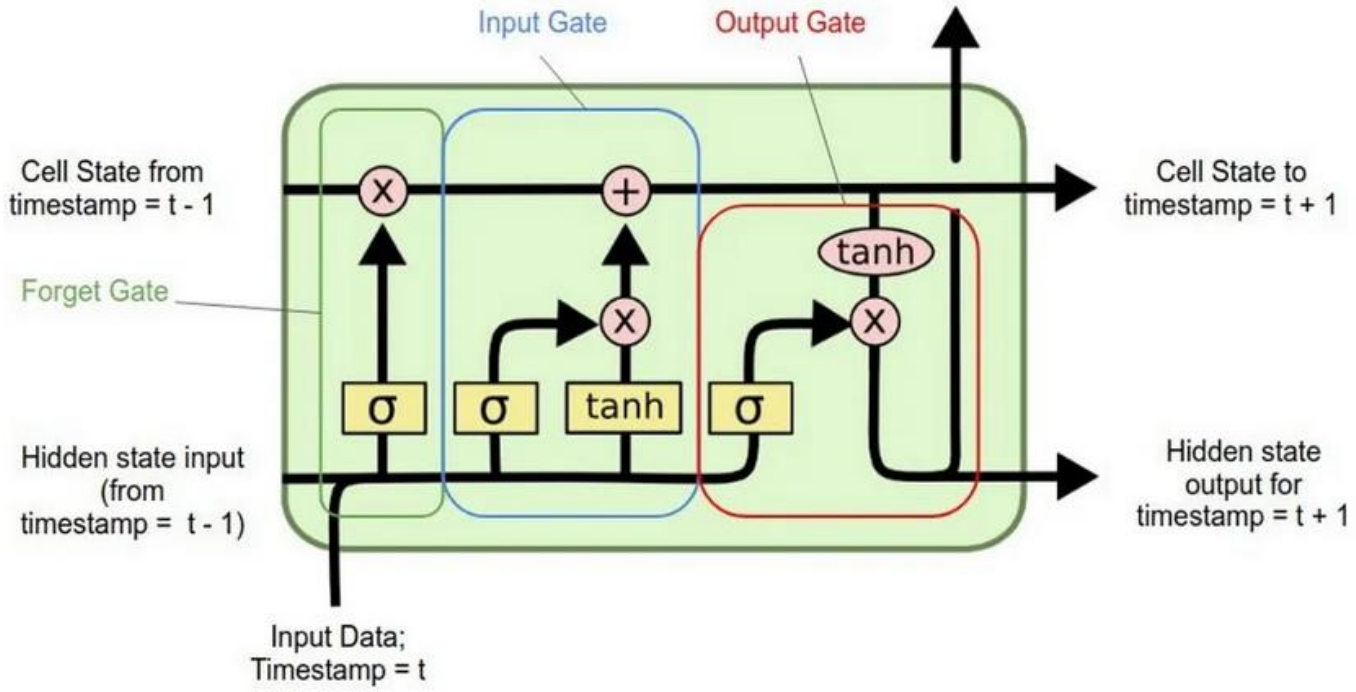
The repeating module in an LSTM contains four interacting layers.



LSTM

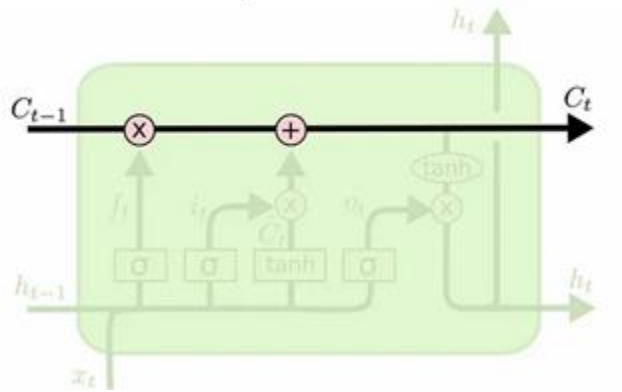
- LSTM → 4 components

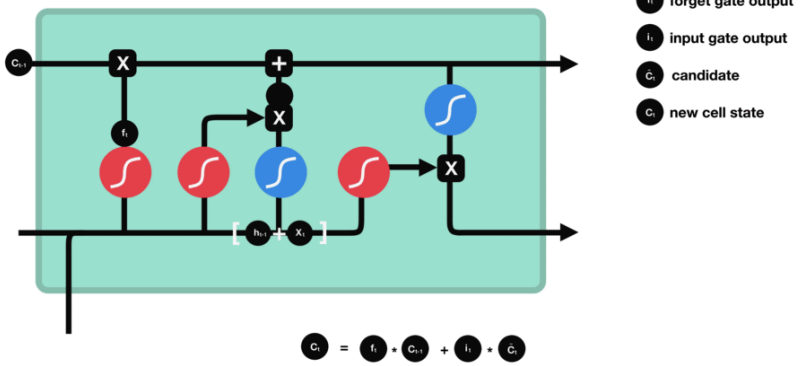
1. Cell state (Memory cell)
2. Forget gate
3. Input gate
4. Output gate



LSTM

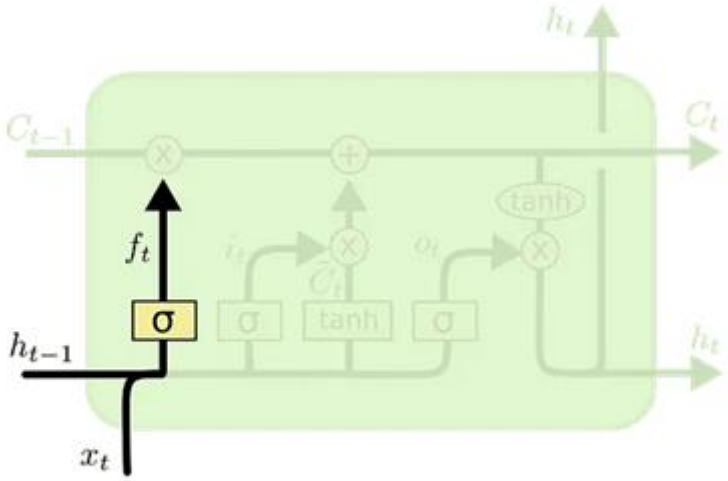
- Cell state (Memory cell) → runs through the entire LSTM unit.
  - It can be thought of as a conveyer belt.
  - Responsible for remembering and forgetting.
  - Some of previous information should be remembered while some of them should be forgotten and some of the new information should be added to the memory.
  - The first operation (**X**) is the pointwise operation which is nothing but multiplying the cell state by an array of [-1, 0, 1]. The information multiplied by 0 will be forgotten by the LSTM.
  - Another operation is (**+**) which is responsible to add some new information to the state.

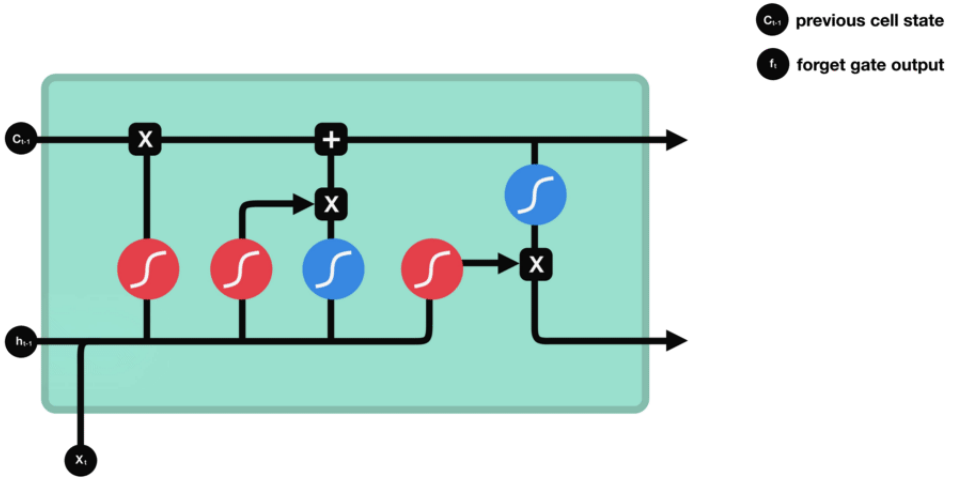




LSTM

- Forget gate
  - decides what information should be forgotten.
  - A sigmoid layer is used to make this decision.
  - This sigmoid layer is called the "forget gate layer"
  - It does a dot product of ***h(t-1)*** and ***x(t)*** and with the help of the sigmoid layer, outputs a number between 0 and 1 for each number in the cell state ***C(t-1).***
  - 0 → forget and 1 → keep



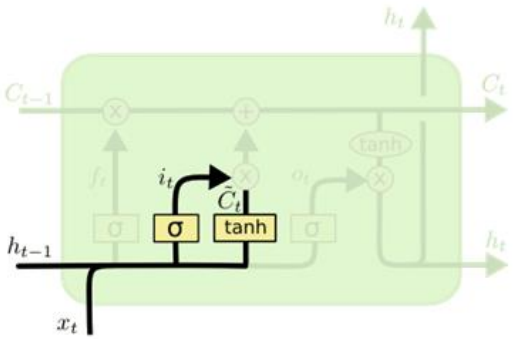


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM

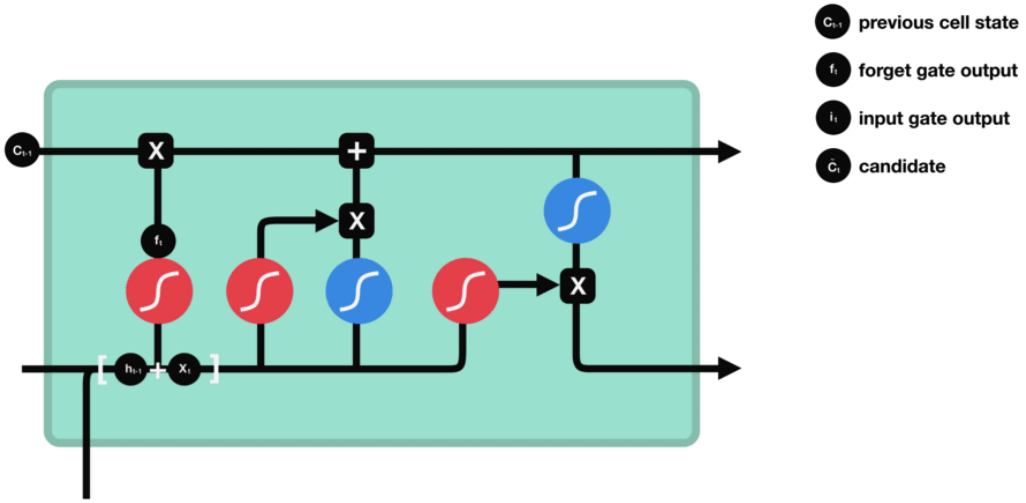
- Input gate
  - gives new information to the LSTM and decides if that new information is going to be stored in the cell state. Composed of 3 parts:

1. A ***sigmoid*** layer decides the values to be updated. This layer is called the "input gate layer"
2. A ***tanh*** activation function layer creates a vector of new candidate values, ***Č(t)***, that could be added to the state.
3. Then we combine these 2 outputs, ***i(t) * Č(t),*** and update the cell state.



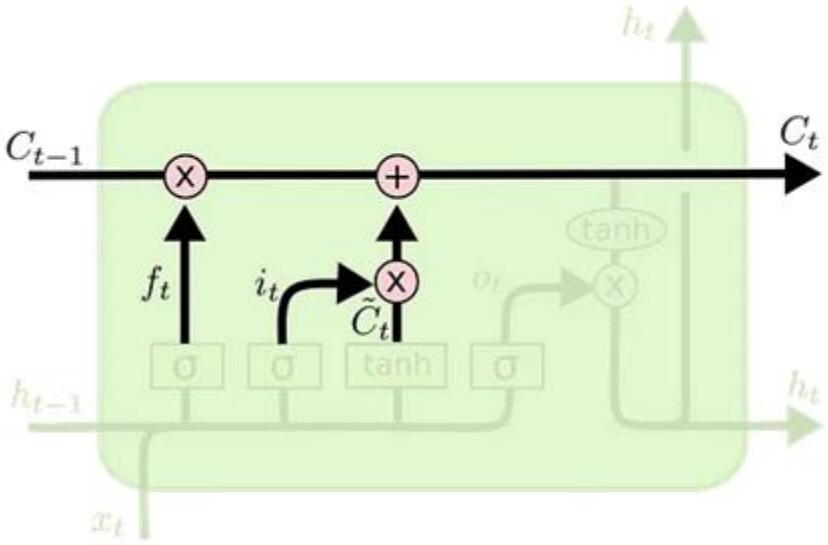
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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



LSTM

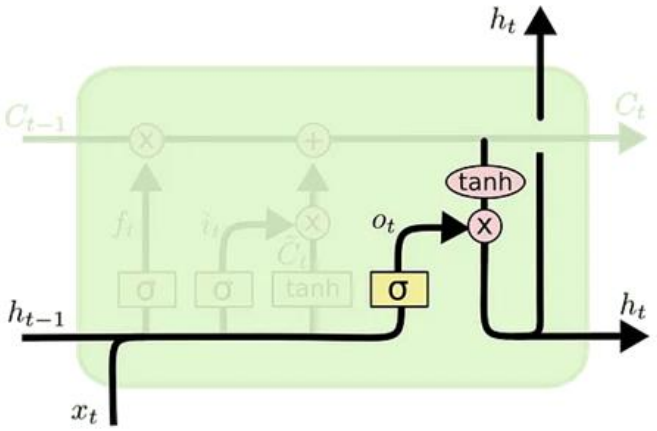
- The new cell state C_t

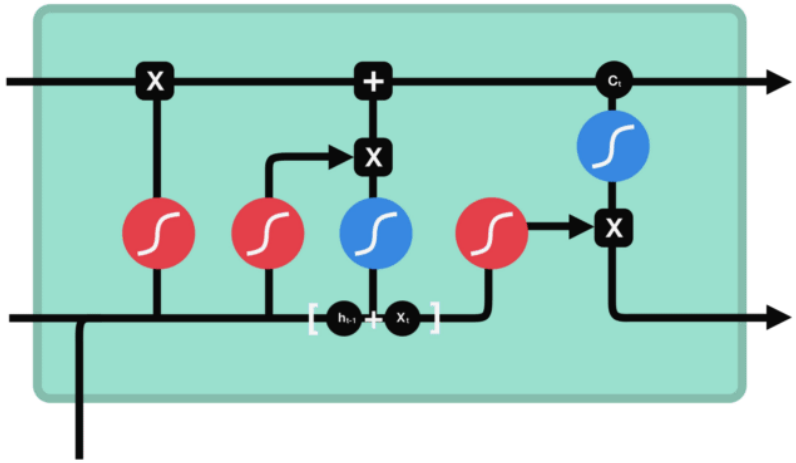


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM

- Output gate
  - depends on the new cell state.
  1. a sigmoid layer decides what parts of the cell state we're going to output.
  2. Then, a ***tanh*** layer is used on the cell state to squash the values between -1 and 1.
  3. finally the values of 1 and 2 are multiplied



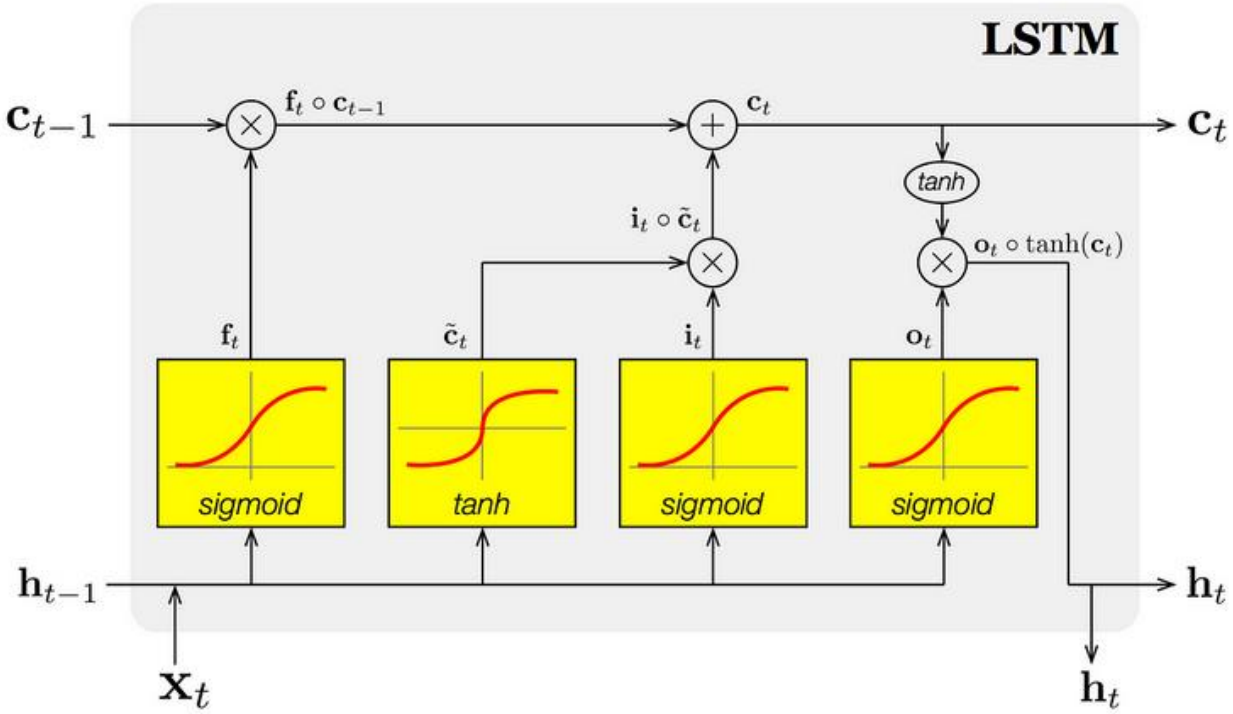


previous cell state
forget gate output
input gate output
candidate
new cell state
output gate output
hidden state

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

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

LSTM



Gating variables

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

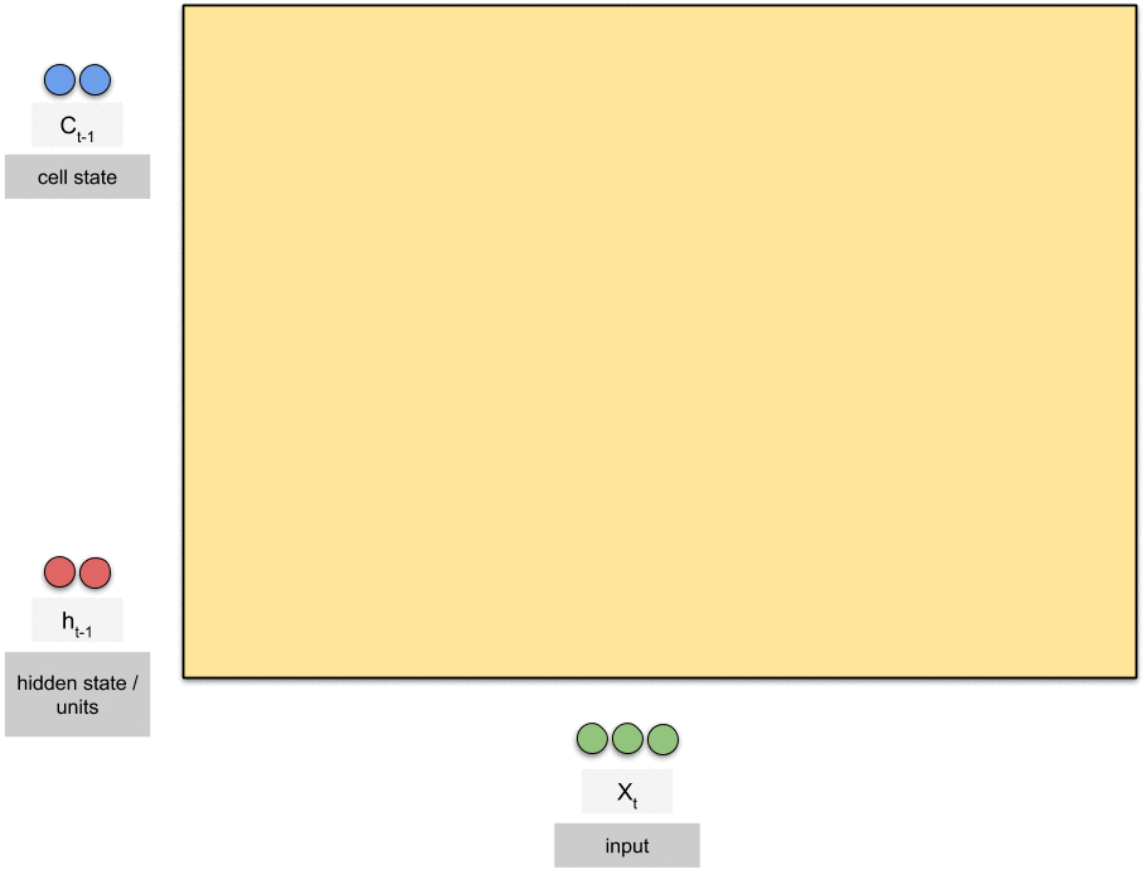
Candidate (memory) cell state

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$

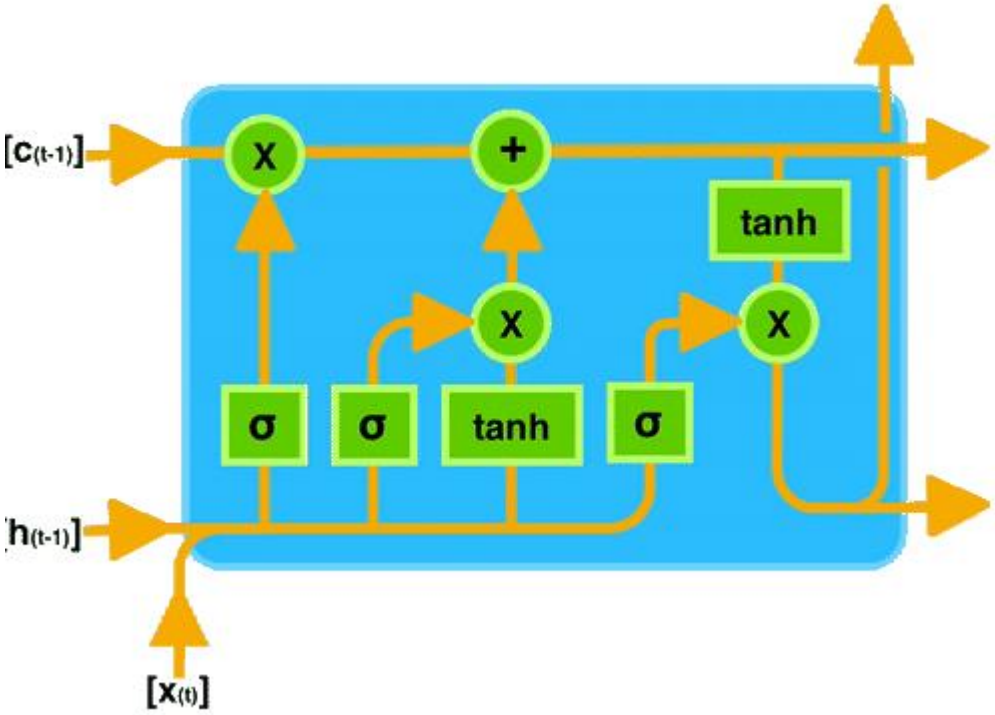
Cell & Hidden state

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

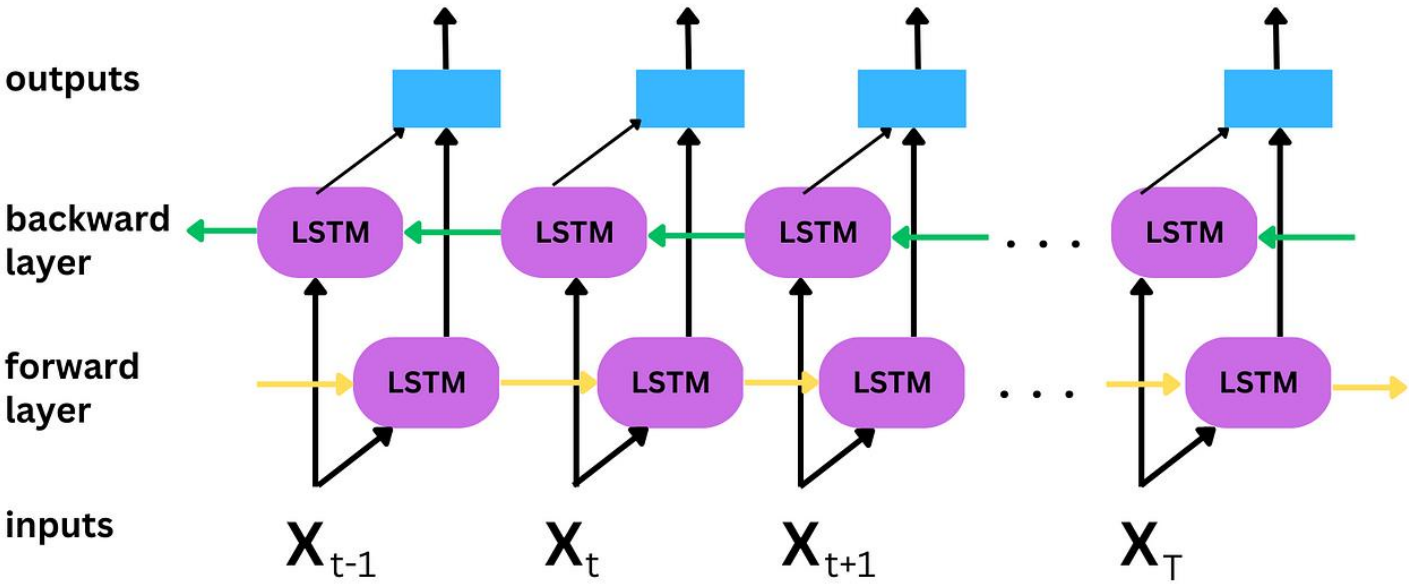
LSTM



LSTM

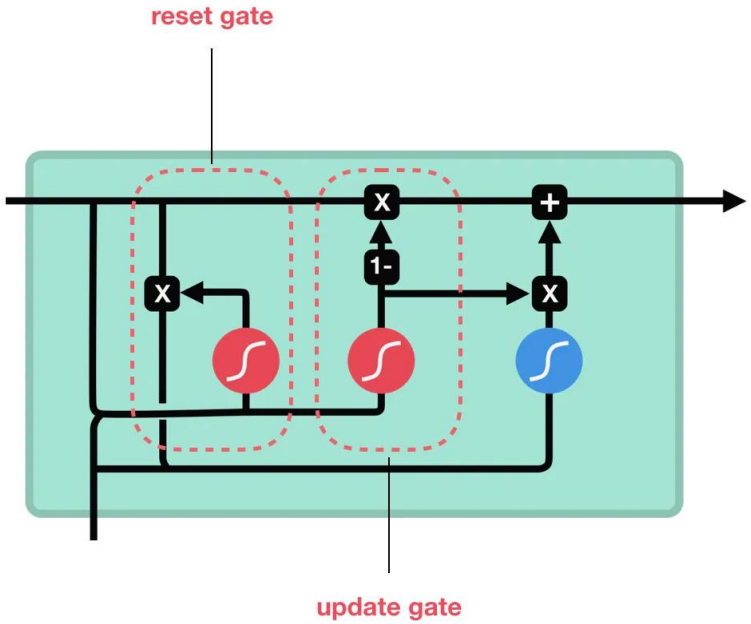


BIDIRECTIONAL LSTM



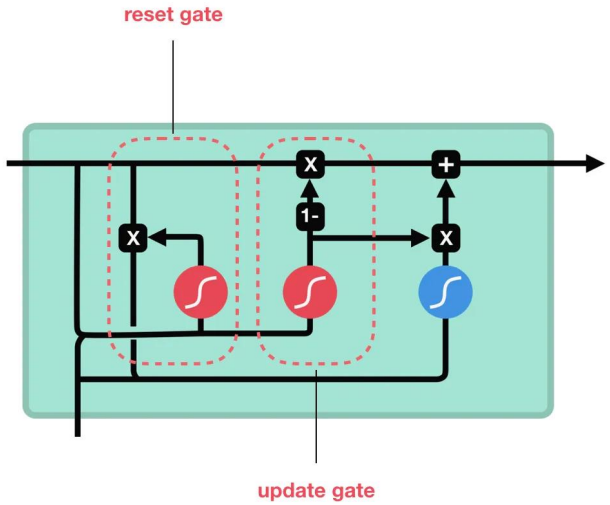
GATED RECURRENT UNIT NETWORK (GRU)

- GRU → newer generation of Recurrent Neural networks
- pretty like LSTM.
- GRU's got rid of the cell state and used the hidden state to transfer information.
- has two gates, a reset gate and update gate.



GATED RECURRENT UNIT NETWORK (GRU)

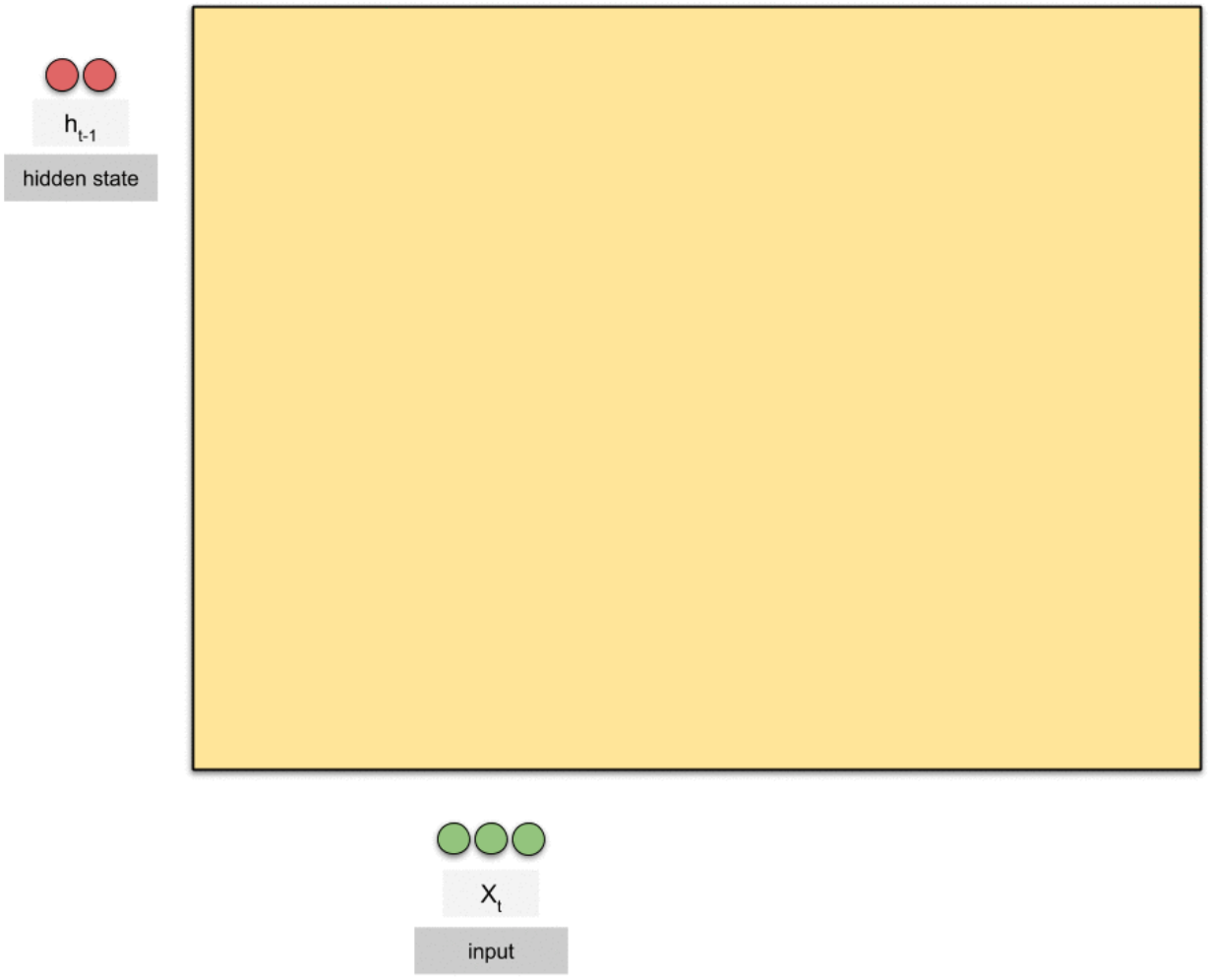
- **Update Gate**
- The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.
- **Reset Gate**
- The reset gate is another gate is used to decide how much past information to forget.



Note:

- GRU's has fewer tensor operations; therefore, they are a little speedier to train then LSTM's.
- There isn't a clear winner which one is better. Researchers and engineers usually try both to determine which one works better for their use case.

ANIMATED GRU CELL

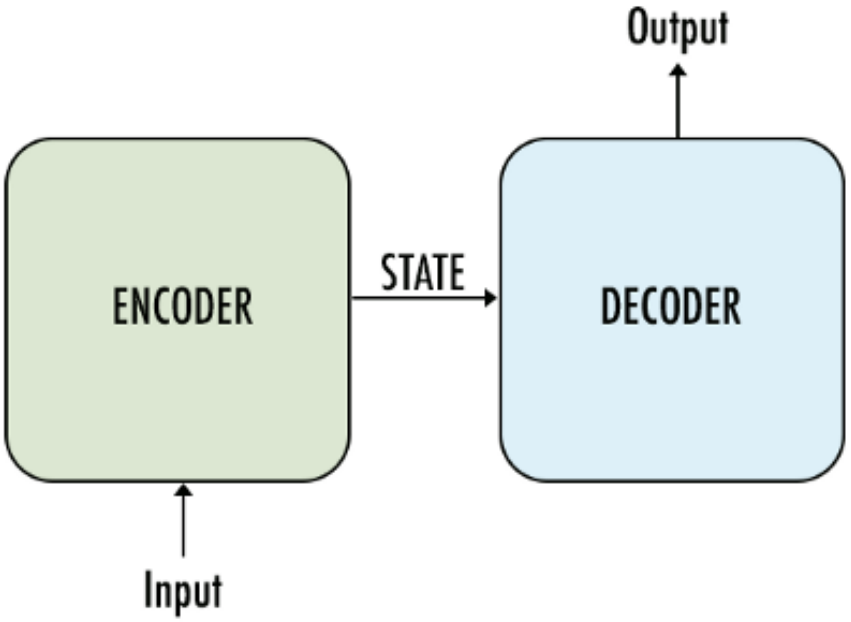


SUMMARY ABOUT RNNS

- RNN's → good for processing sequence data for predictions but suffers from short-term memory.
- LSTM's and GRU's → created to mitigate short-term memory using mechanisms called gates.
- Gates → neural networks that regulate the flow of information flowing through the sequence chain.
- LSTM's and GRU's → used in state of the art deep learning applications like speech recognition, speech synthesis, natural language understanding, etc.

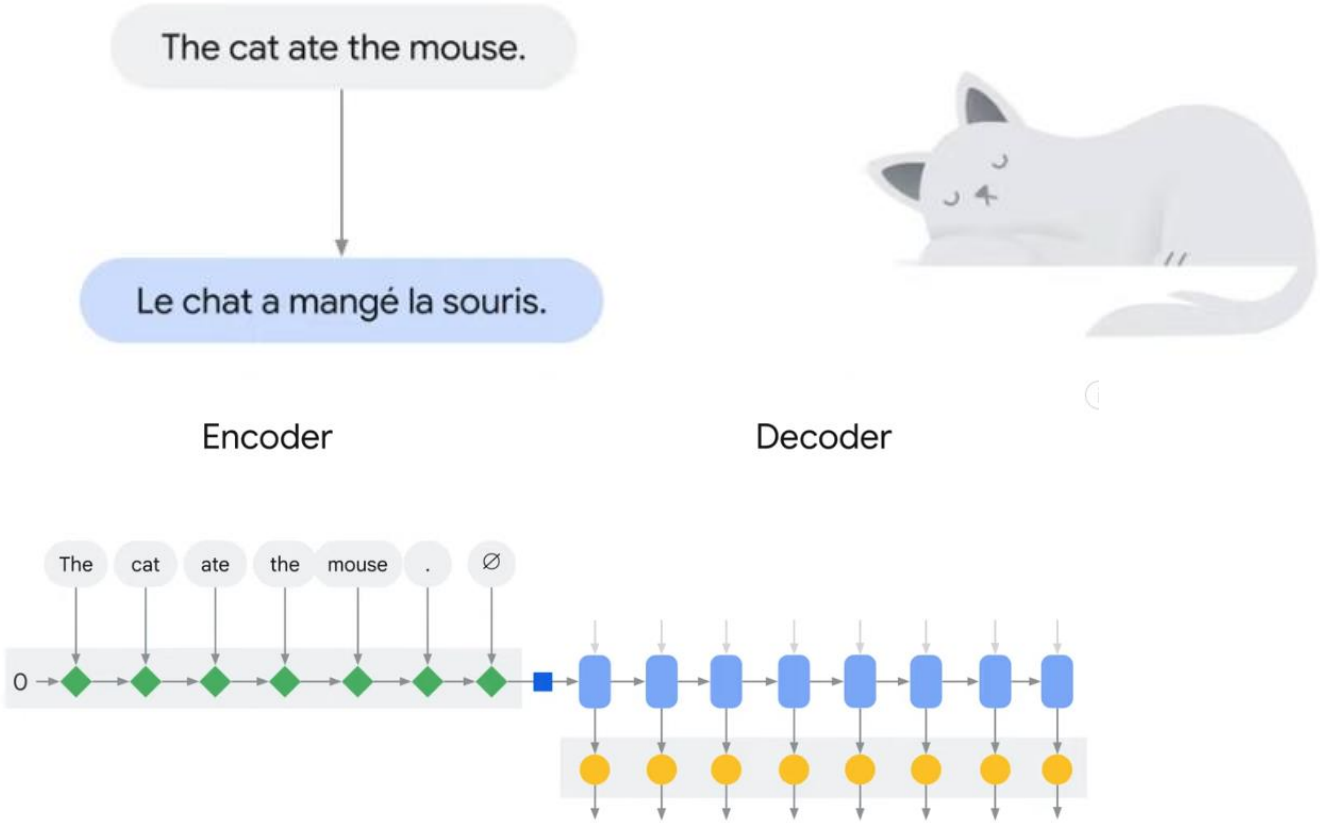
ENCODER-DECODER RNN

- Sequence to sequence model
- Implemented by many-to-many RNN
- Composed of two blocks, one for encoding and the other for decoding



ENCODER-DECODER RNN

- Encoder-decoder architectures can be applied to various sequence-to-sequence tasks, including machine translation, text summarization, speech recognition, and more



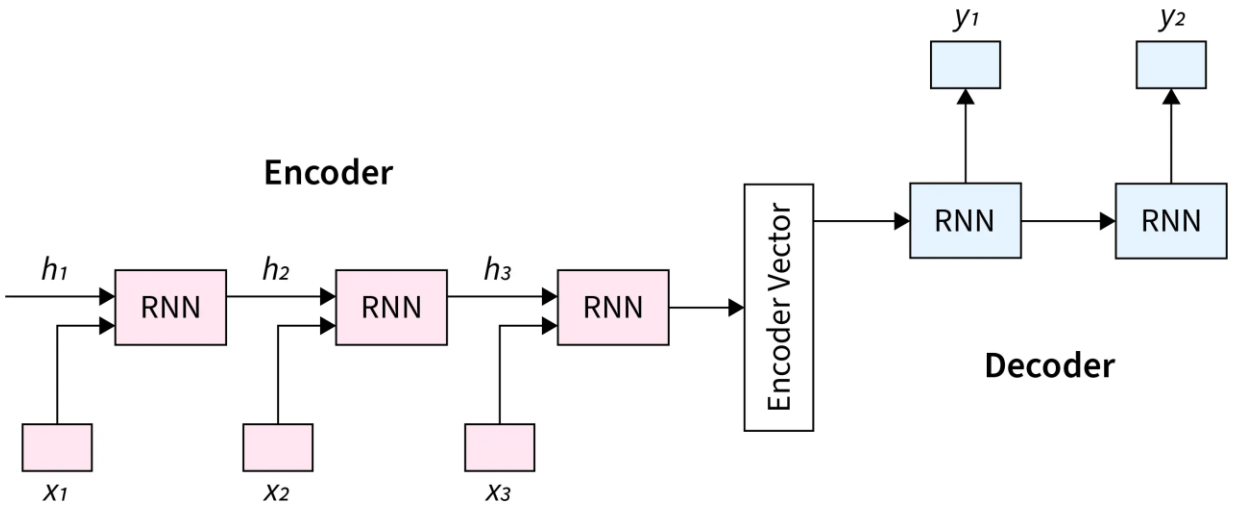
ENCODER-DECODER RNN

- **Encoder:**
  - The encoder takes in the input sequence and processes it into a fixed-size context or hidden state.
  - It sequentially processes each element of the input sequence and updates its internal hidden state at each time step.
  - The final hidden state of the encoder serves as a representation or summary of the entire input sequence.
  - Commonly, recurrent neural networks (RNNs) or other sequence-based models are used as encoders.
    - LSTMs or GRUs are intensively used.

ENCODER-DECODER RNN

- **Decoder:**
  - The decoder takes the context or hidden state produced by the encoder and generates the output sequence.
  - It is designed to predict the next element in the sequence at each time step.
  - Like the encoder, the decoder is often implemented using RNNs or similar recurrent structures (LSTM, GRU, …).
  - The decoder's initial hidden state is set to the final hidden state of the encoder.
  - During training, the correct output sequence is provided to the decoder, and the model is optimized to minimize the difference between predicted and actual outputs.

PUTTING THEM TOGETHER

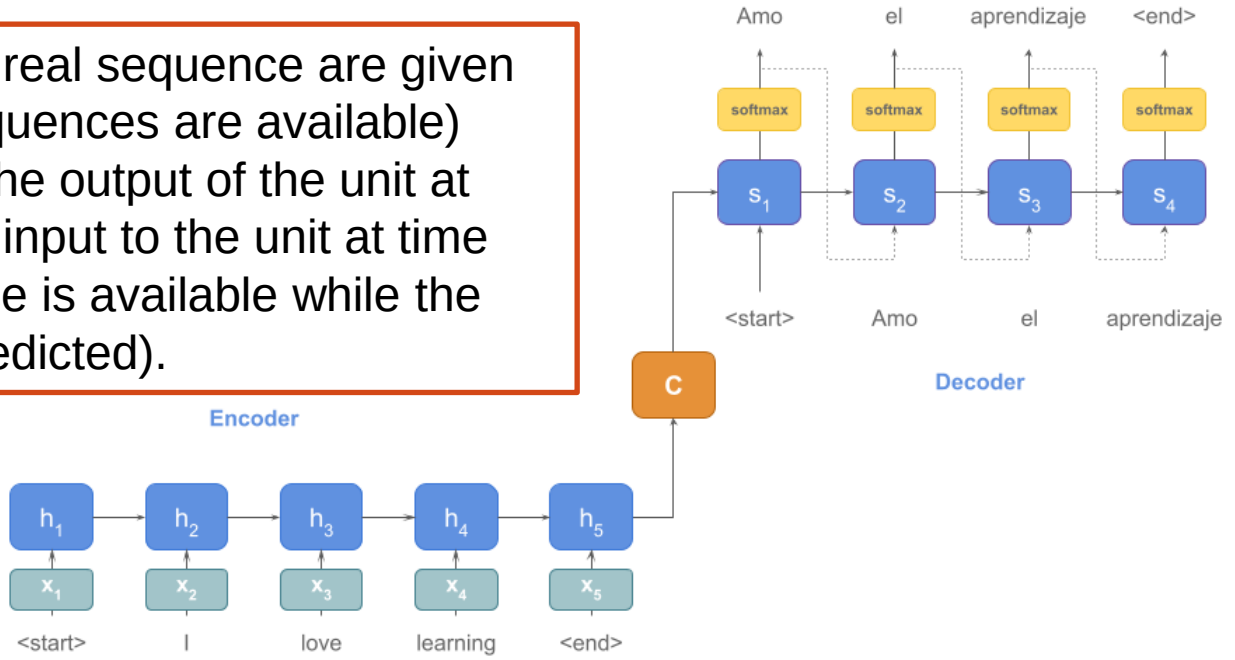


SCALER Topics

TRAINING ENCODER-DECODER

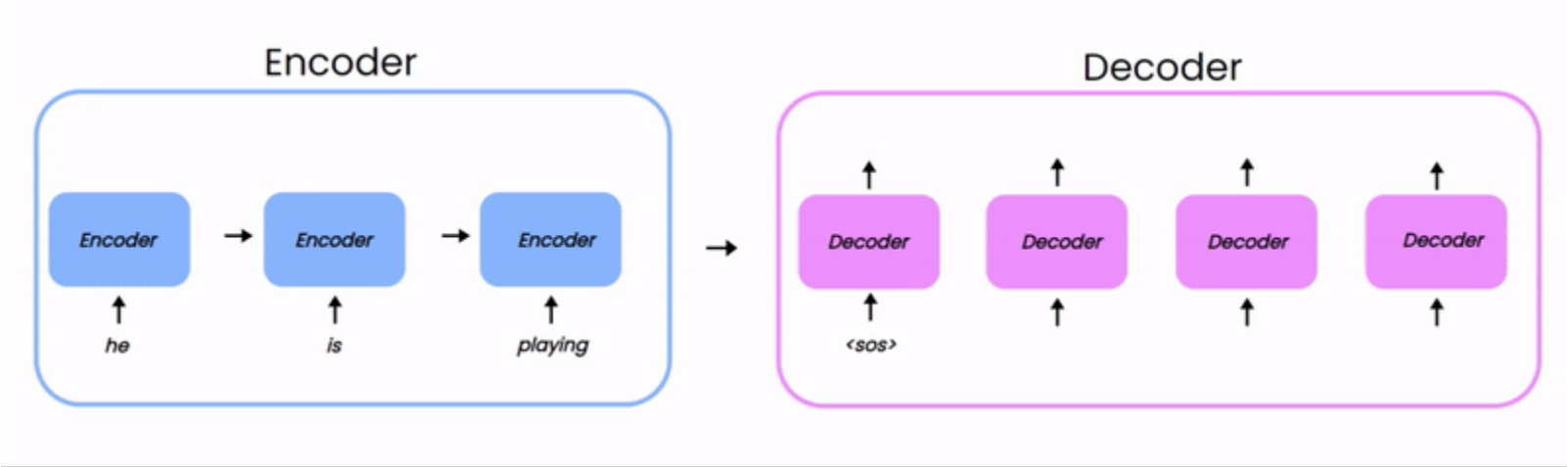
- **Training:**
  - The model is trained using a dataset of input-output sequence pairs.
  - The encoder processes the input sequence, and its final hidden state is passed to the decoder.
  - The decoder generates the output sequence one element at a time.
  - The training objective is to minimize the difference between the predicted and target output sequences.

- **During training**, the real sequence are given to decoder (the 2 sequences are available)
- **During prediction**, the output of the unit at time t is given as the input to the unit at time t+1 (the first sequence is available while the second should be predicted).



MORE TO KNOW

- Always, all the sequences to be predicted are augmented with a special start and end symbols at the start of the sequence.
  - "start-of-sequence" (SOS) and "end-of-sequence" (EOS) tokens.
  - SOS is given as input with the data coming from the encoder to start the prediction.
  - EOS helps the network to identify the end of the predicted sequence.



ADVANTAGES

1. **Versatility in Sequence-to-Sequence Tasks:**
   - Encoder-decoder architectures are versatile and can be applied to a wide range of sequence-to-sequence tasks, including machine translation, text summarization, speech recognition, and more.
2. **Handling Variable-Length Input and Output Sequences:**
   - Encoder-decoder models are designed to handle input and output sequences of variable lengths. This makes them suitable for tasks where the length of the input and output may vary.
3. **Information Compression:**
   - The encoder compresses the information from the input sequence into a fixed-size context or hidden state, which serves as a summary of the input. This allows the model to capture essential information and discard unnecessary details.
4. **Effective for Tasks with Temporal Dependencies:**
   - Encoder-decoder architectures, especially those with recurrent structures, are effective for tasks involving temporal dependencies, where the order of elements in the sequence matters.

DISADVANTAGES

1. **Limited Memory:**
   - Traditional encoder-decoder architectures, especially those relying solely on recurrent layers, may struggle with capturing long-term dependencies due to vanishing or exploding gradient problems. Long sequences may result in information loss over time.
2. **Inability to Handle Global Context Efficiently:**
   - In some cases, the context vector produced by the encoder may not efficiently capture global information from the entire input sequence, especially if the input sequence is very long. This limitation can impact the model's understanding of the context.
3. **Training Complexity:**
   - Training encoder-decoder models can be computationally intensive and time-consuming, especially for large datasets and complex tasks. Training can also be challenging when handling variable-length sequences.
4. **Sensitivity to Hyperparameters:**
   - The performance of encoder-decoder models may be sensitive to hyperparameter choices, such as learning rates, architecture design, and the choice of attention mechanisms. Suboptimal hyperparameter tuning may affect convergence and overall performance.
5. **Difficulty in Handling Rare Sequences:**
   - Encoder-decoder models may struggle with generating rare or out-of-vocabulary sequences, as the training data may not provide sufficient examples for such cases.

ATTENTION MECHANISM

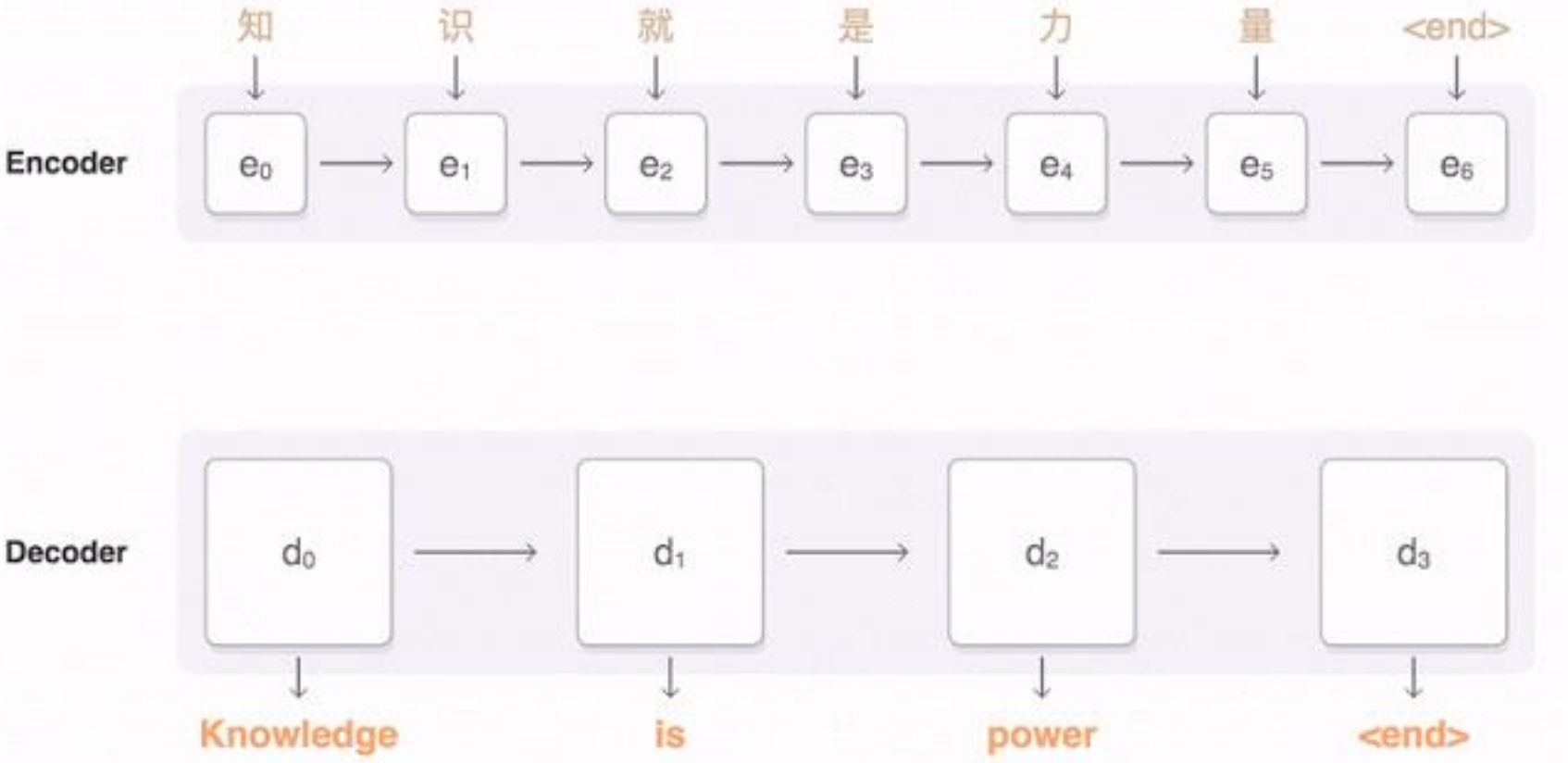
- RNNs → used successfully for many tasks involving sequential data such as machine translation, sentiment analysis, image captioning, time-series prediction etc.
- Improved RNN models (LSTMs, GRUs, …) → enable training on long sequences overcoming problems like vanishing gradients.
  - However, even the more advanced models have their limitations working with long data sequences
  - Performance drops for longer sentences
- In machine translation, for example, the RNN has to find connections between long input and output sentences composed of dozens of words.
- Existing RNN architectures needed to be changed and adapted to better deal with such tasks.
- **SOLUTION**: **Attention Mechanism**

ATTENTION MECHANISM

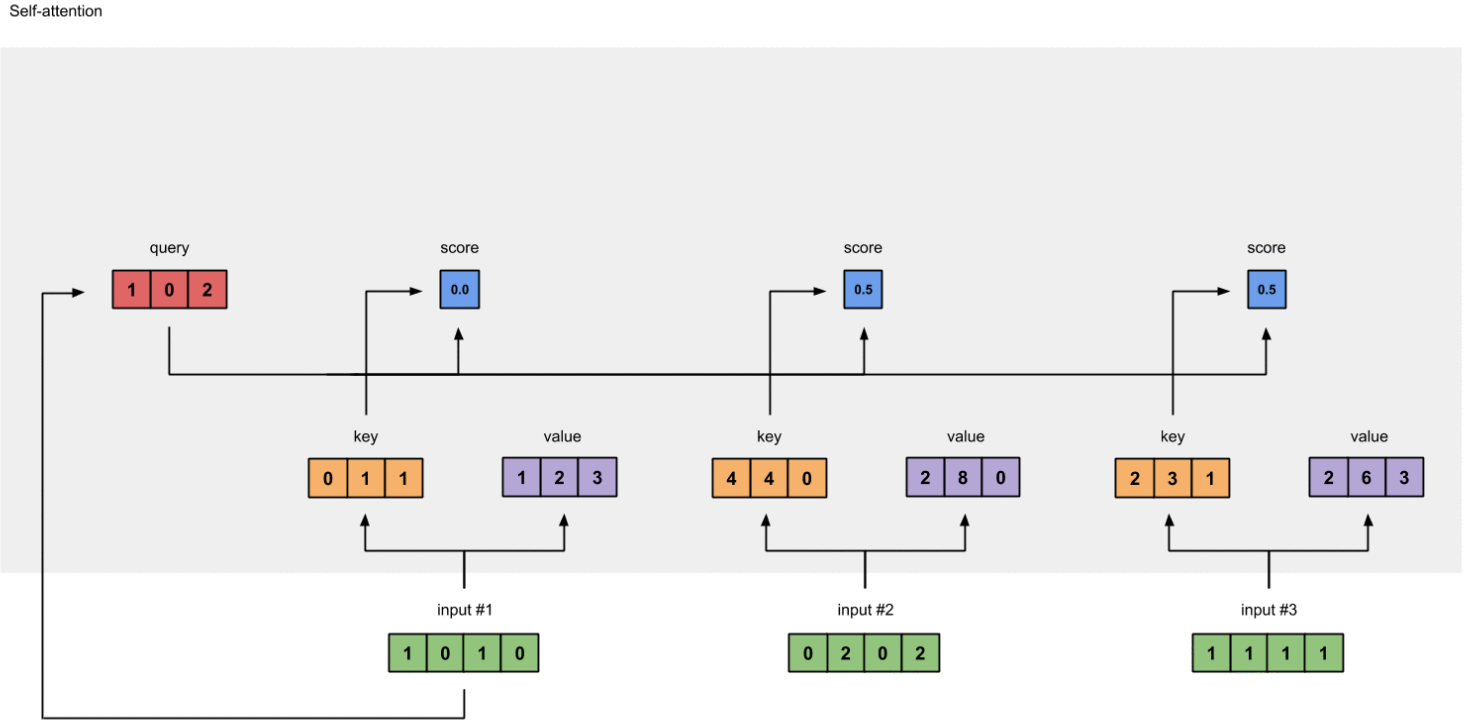
- Attention is a mechanism combined in the RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and of higher quality.
- **Context Vector:**
  - context vector of fixed-size provided by the encoder and passed to decoder.
- **Attention Weights**:
  - Instead of relying solely on a fixed-size context vector, attention mechanisms introduce a set of attention weights.
  - are dynamically calculated for each element in the input sequence based on its relevance to the current step in the decoding process.
  - calculated using a scoring mechanism that measures the similarity or relevance between the current decoder hidden state and each encoder hidden state.
  - Common scoring mechanisms include dot product, additive attention, and multiplicative attention.

ATTENTION MECHANISM

As general overview, the output of each hidden unit with a score is fed into all the decoder units with the normal input.

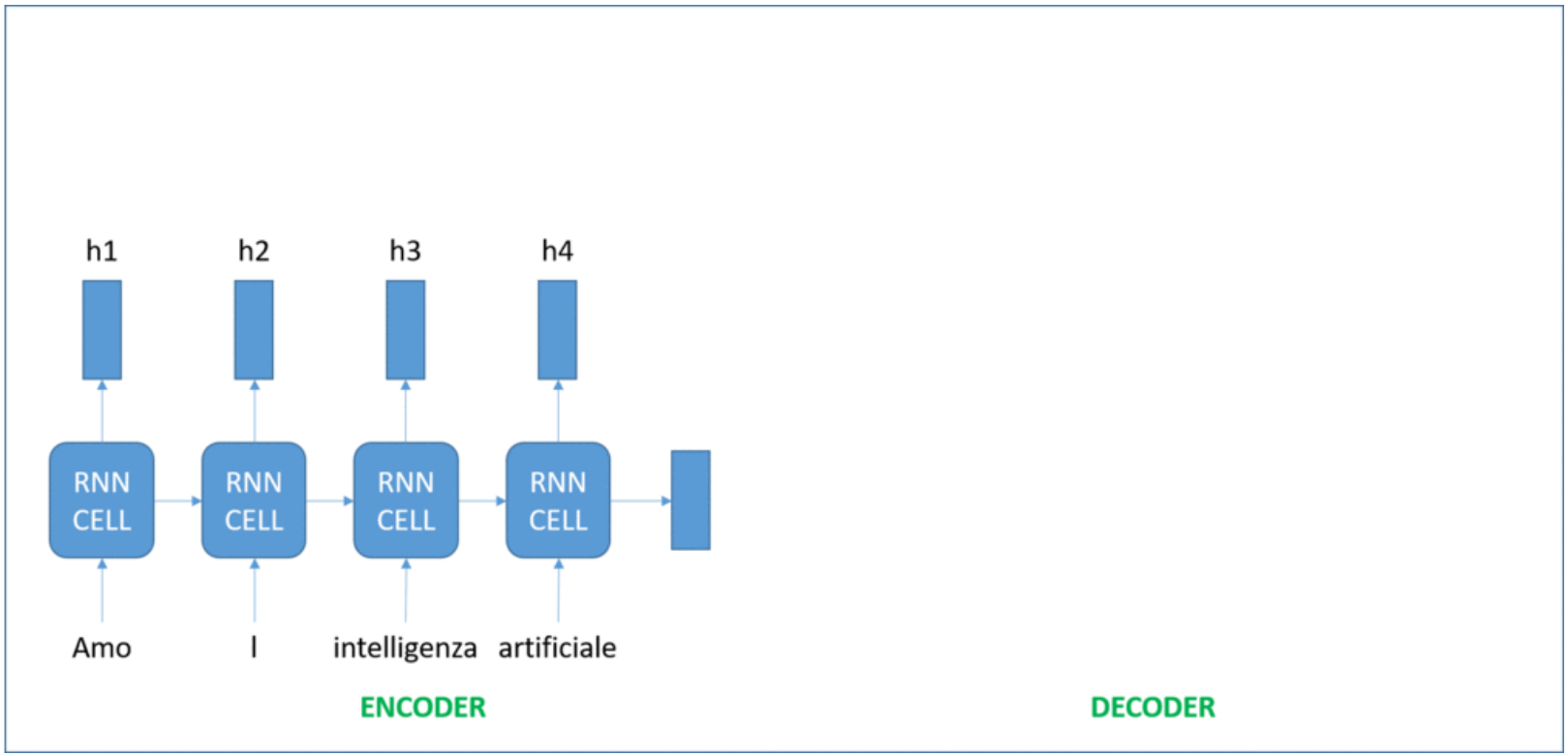


SELF-ATTENTION



ATTENTION MECHANISM

- **Weighted Sum of Encoder Hidden States:**
  - The attention weights are used to compute a weighted sum of the encoder hidden states that is used in combination with the current decoder hidden state to make predictions for the current time step.



ATTENTION MECHANISM