Understanding LLMs

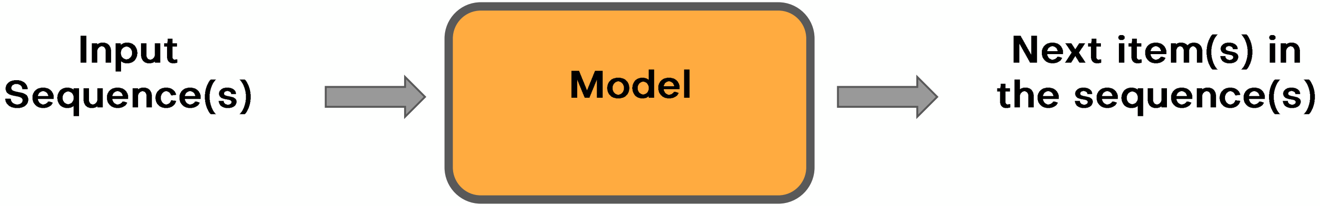
OUTLINE

- Sequential Data Problems
- Intro to LLMs
- Overview of RNNs
- Architecture of RNNs
- Working Principles of RNNs
- Limitations of Traditional RNNs
- Different Variants of RNNs
- Alternative Solution to Traditional RNNs

Sequential Data Problems

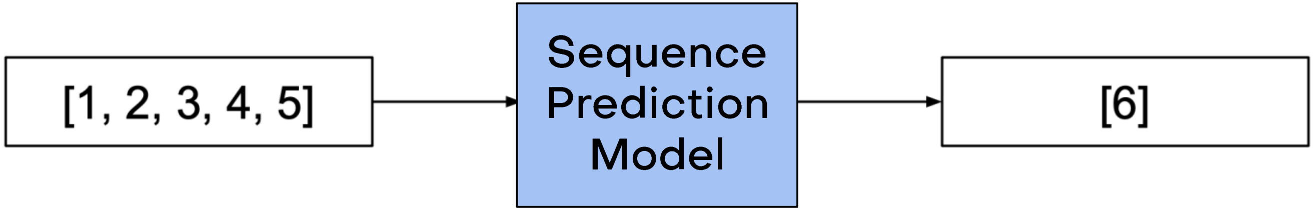
Process Data as a Sequence

- Sequence prediction is different from other types of supervised learning problems.
- The sequence imposes an order on the observations that must be preserved when training models and making predictions.
- Generally, prediction problems that involve sequence data are referred to as sequence prediction problems.



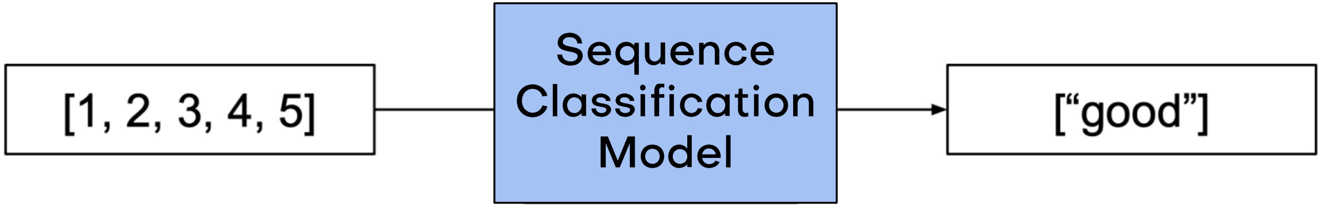
Sequence Prediction Problems

- **Weather Forecasting.** Given a sequence of observations about the weather over time, such as previous temperature measurements, predict the expected temperature tomorrow.



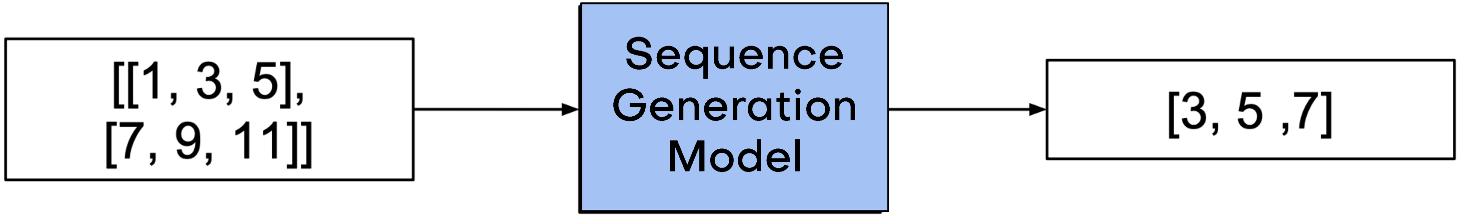
Sequence Classification Problems

- **Sentiment Analysis.** Given a sequence of text such as a review or a tweet, predict whether the sentiment of the text is positive or negative.



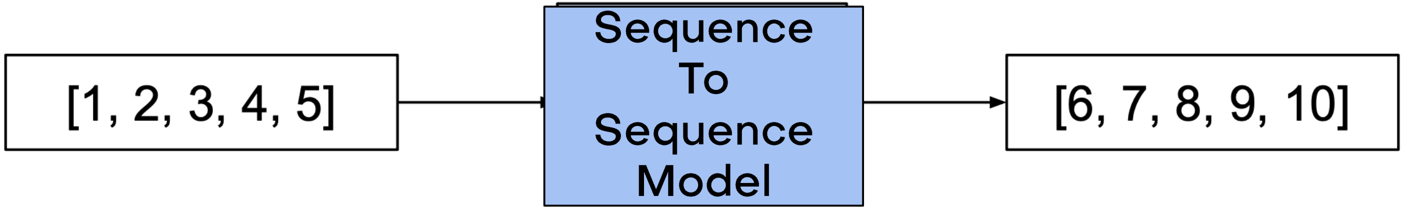
Sequence Generation Problems

- **Text Generation.** Given a corpus of text, generate new sentences or paragraphs of text that read like they could have been drawn from the corpus.



Sequence to Sequence Prediction

- **Text Summarization.** Given a document of text, predict a shorter sequence of text that describes the salient parts of the source document.



Modeling Sequential Data – Text

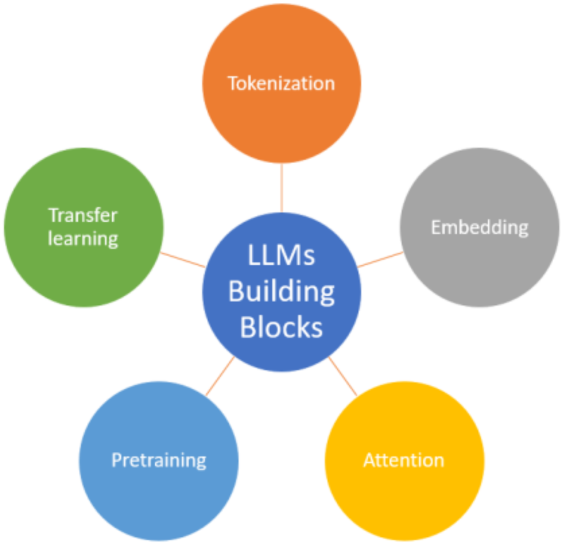
1. **Multi-layer feed-forward Neural Network** is only meant for data points, which are independent of each other.
2. **Convolutional Neural Network** is used for images and 2 dimensional data
3. **Recurrent Neural Network (RNN)**
4. **Large Language Models (LLMs)**

Intro to LLMs

What are Large Language Models (LLMs)?

- **LLM** is a deep learning algorithm that is capable of recognizing, summarizing, translating, predicting and generating text and other forms of content by leveraging insights extracted from vast datasets.

- LLMs are among the most successful applications of transformer models.



What does “Large” in LLM mean?

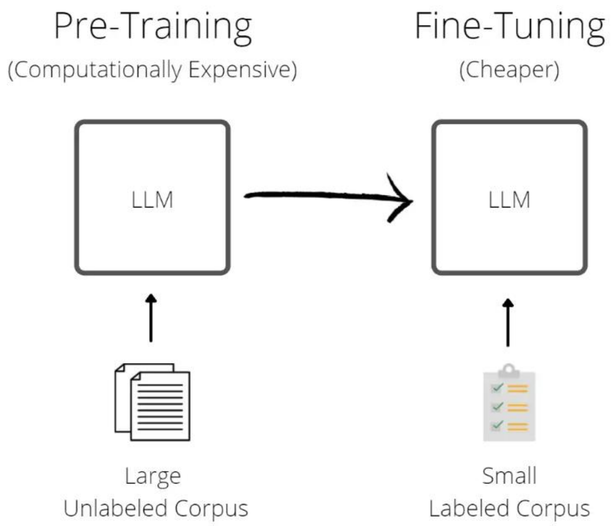
The term "large" in LLM indicates the vast number of parameters the model can autonomously modify during its learning process. Several highly successful LLMs encompass hundreds of billions of parameters.

Large Language Model

Why LLMs Are Important?

- LLMs advance natural language processing, improving accuracy across tasks through pre-training on extensive data and fine-tuning.

- LLMs process vast data, boosting prediction and classification accuracy, while accurately representing language, generating text, and enhancing models.



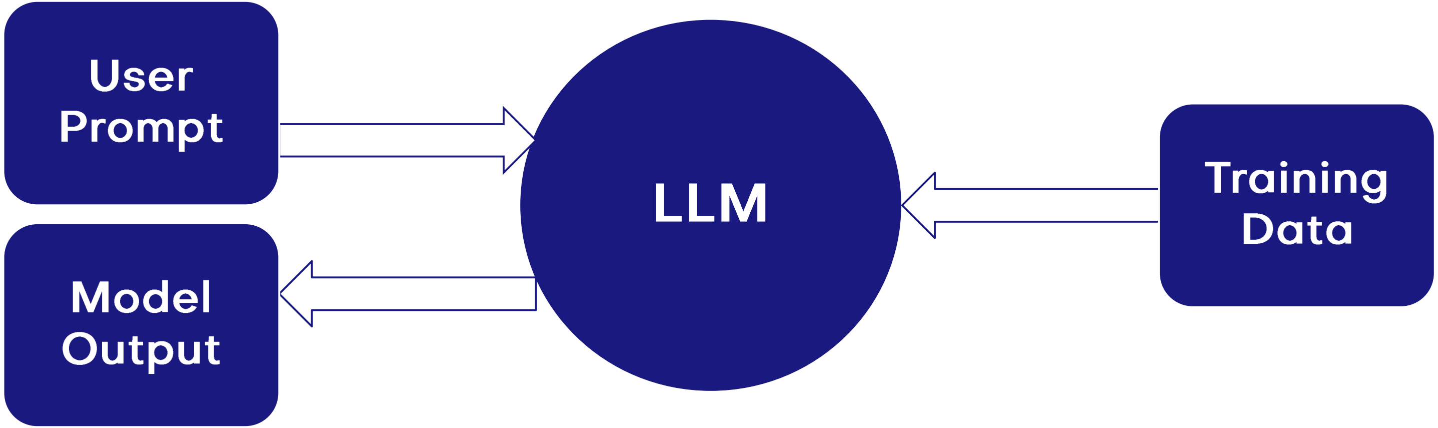
Role of Language Models in NLP Tasks

- Machine Translation
- Sentiment Analysis
- Question Answering
- Summarization & Classification
- Content Creation
- Text Generation

Some of the Most Popular LLMs

- GPT-3
- BERT
- XLNet
- T5
- RoBERTa

How are LLMs Trained?

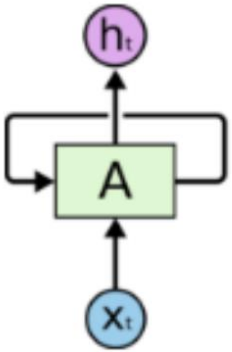


Overview of Recurrent Neural Networks (RNNs)

What are RNNs?

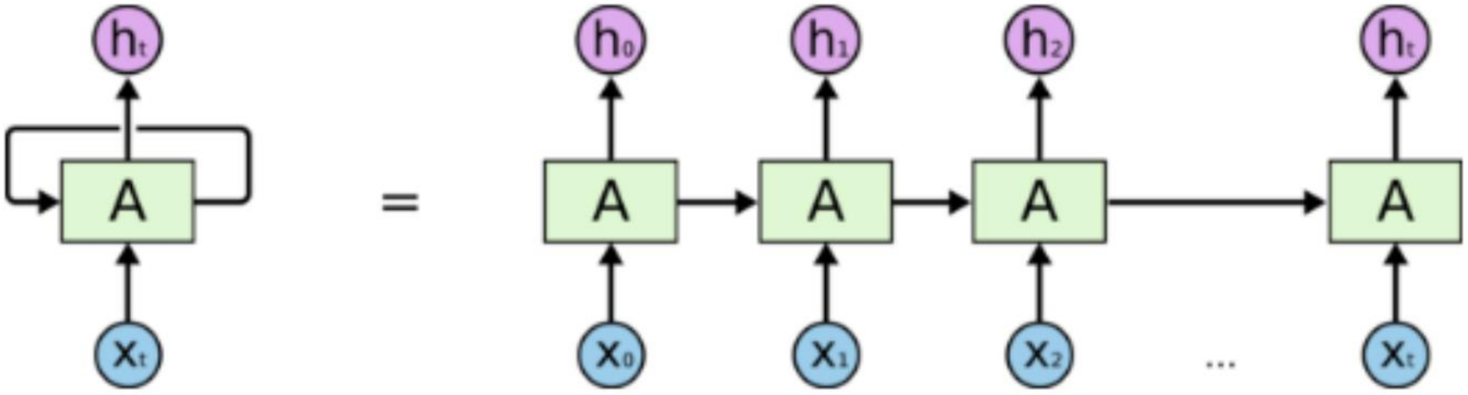
- **RNNs** are specialized neural networks designed to process sequential data like text.

- They consider dependencies between data points and employ '**memory**' to utilize previous information for generating the next output.



Unrolling an RNN

- An RNN has a feedback loop that can be unrolled k time steps.

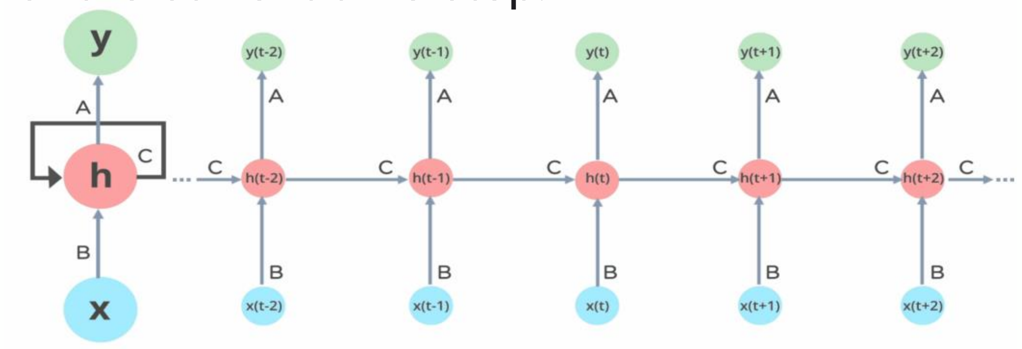


An unrolled recurrent neural network.

Unrolling an RNN

Each time step takes two inputs:

- The output of the network from the previous time step is used as input.
- The internal state from the previous time step serves as a starting point for the current time step.

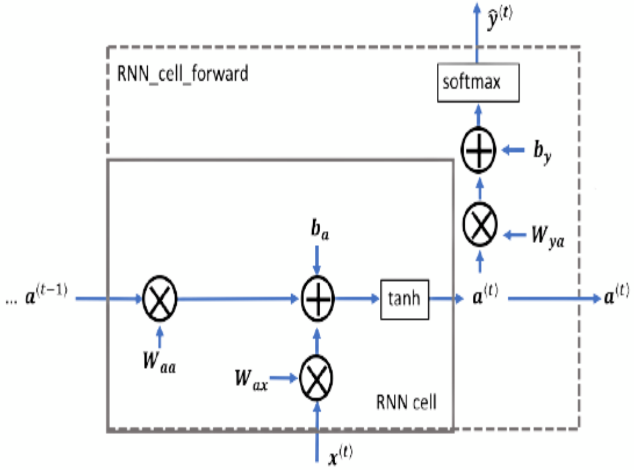


Basic RNN Cell Mathematical Formulation

- **RNN receives two inputs**: "a" (previous hidden state) at time step t-1 and "x" at time step t.

- **Compute the weighted sum** of "a" and "x" and pass it through an activation function (tanh in the figure).

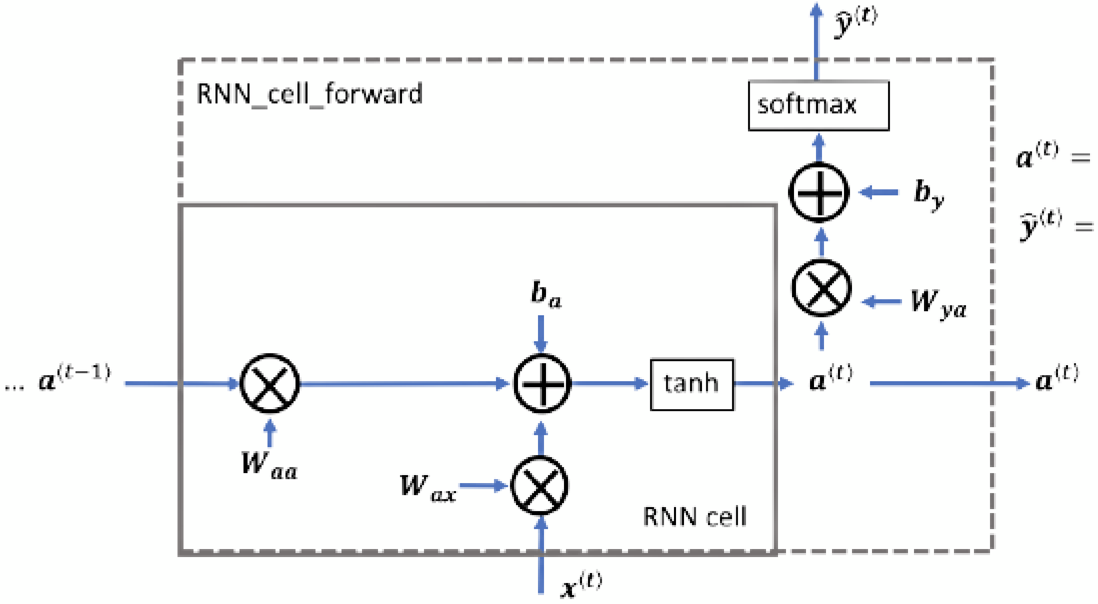
- **Activation function generates two outputs** for the RNN:
    - "a_t": Hidden state passed to the next cell after tanh activation.
    - "y_t": Final output obtained by passing "a_t" through a softmax activation.



SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

أكاديمية سدايا
SDAIA Academy

Basic RNN Cell Mathematical Formulation



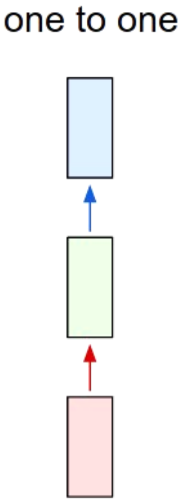
$$a^{\langle t \rangle} = \tanh(W_{ax}\, x^{\langle t \rangle} + W_{aa}\, a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = softmax(W_{ya} a^{\langle t \rangle} + b_y)$$

Types of RNNs

One-to-One

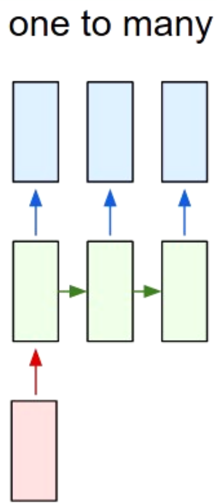
- The simplest form of RNNs; also called Plain Neural networks.
- Processes fixed-size input to fixed-size output, where they are independent of previous information/output.
- Example: Image classification.



One-to-Many

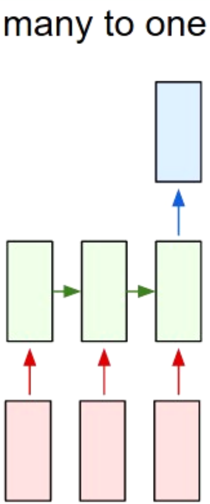
- Processes fixed-size input to generate a sequence of data as output.

- Example: Image Captioning takes the image as input and outputs a descriptive sentence of words.



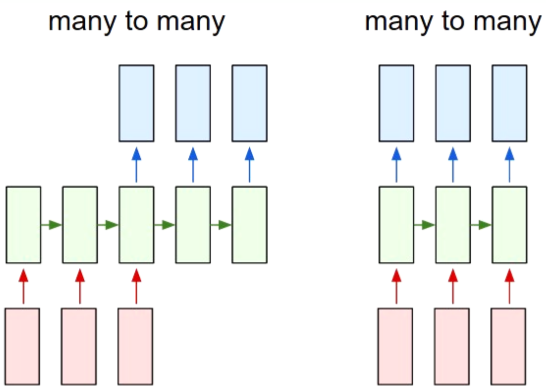
Many-to-One

- Processes a sequence of information as input, producing a single fixed-size output vector.
- Enables RNNs to summarize sequential data into a concise representation.
- Example: Sentiment analysis, where sentences are classified as expressing positive or negative sentiment.



Many-to-Many

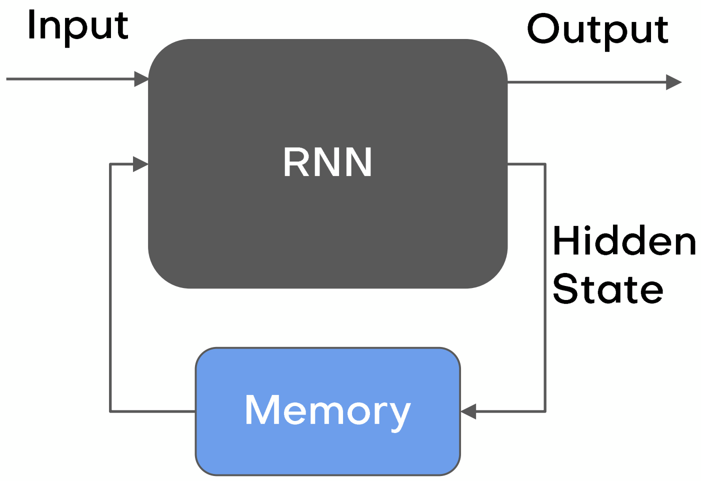
- Processes a sequence of information as input and recurrently outputs a sequence of data.
- A powerful tool for tasks requiring input-output alignment in sequential data, enabling RNNs to transform sequences between different domains or perform structured data classification.
- Example: Machine Translation and Named Entity Recognition (NER)



Architecture of RNNs

Overview

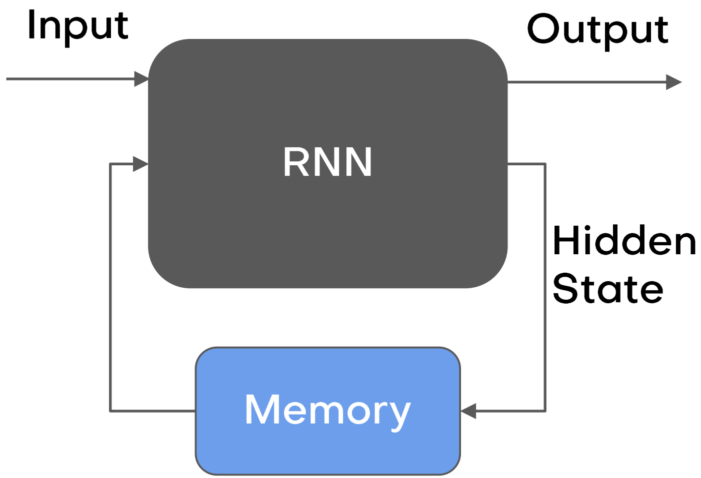
- RNN consists of input, hidden, and output layers with interconnected weights and biases.
- Can be unidirectional (processing input forward) or bidirectional (past and future information).
- Recurrent connection facilitates information sharing across time steps or sequence elements.
- At each time step, RNN takes input and previous hidden state to generate output and update hidden state.



Inputs

- RNNs take sequential data as inputs, which can be represented as a sequence of vectors, such as words in a sentence.

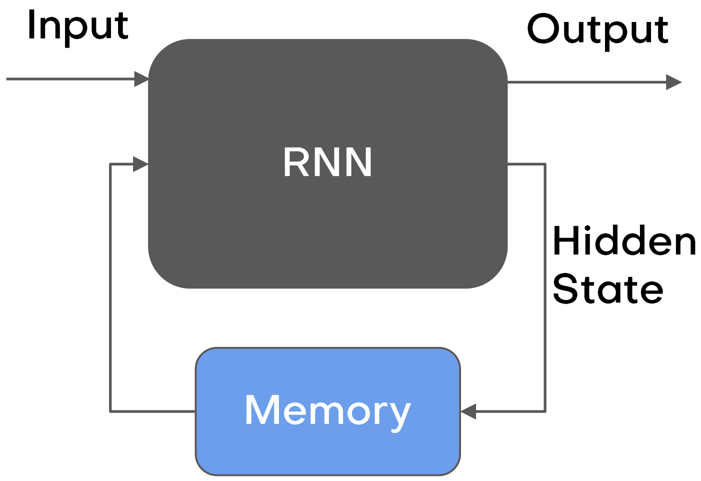
- At each time step, an RNN receives an input vector that represents the current element of the sequence being processed.



Outputs

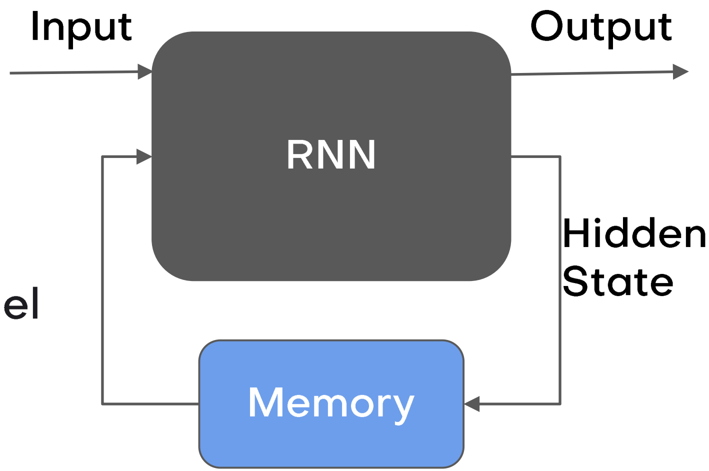
- RNNs produce outputs at each time step, which can be used for different purposes depending on the specific task.

- For sequence prediction tasks, such as language modeling or next word prediction, the output at each time step can be a probability distribution over possible values or classes.



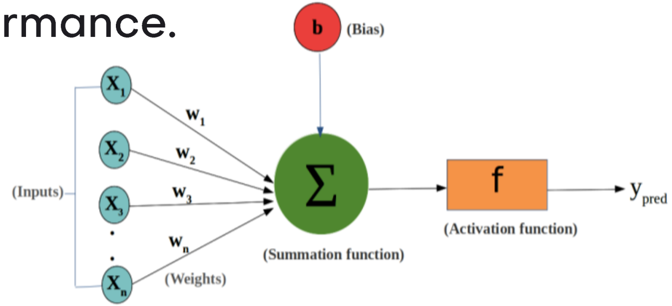
Hidden States

- Hidden states serve as memory or representation of past information.
- Updated at each time step, incorporating current input and previous hidden state.
- Hidden state captures dependencies and contextual information, enabling RNN to model sequential patterns and long-term dependencies.



Weights & Biases

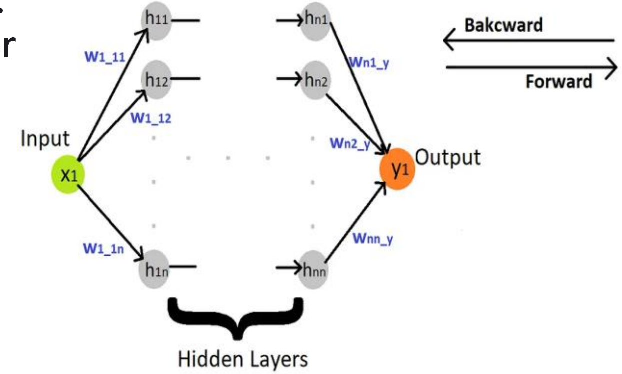
- RNNs have interconnected layers with weights and biases.
- Weights determine the strength of connections between neurons in different layers.
- Biases act as adjustable parameters, introducing flexibility and fine-tuning to the model.
- During training, RNNs learn optimal values for weights and biases to minimize errors and improve performance.



Working Principles of RNNs

Forward Propagation

- Hidden state initialized as memory with zeros or random values.
- RNN processes input and previous hidden state at each step.
- Input and previous hidden state combined for context.
- Combined input passed through activation function for non-linearity.
- Activation output becomes an updated hidden state.
- Iterative process maintains context across the sequence.
- Encodes information from the entire sequence.
- RNN may produce an output vector for predictions.

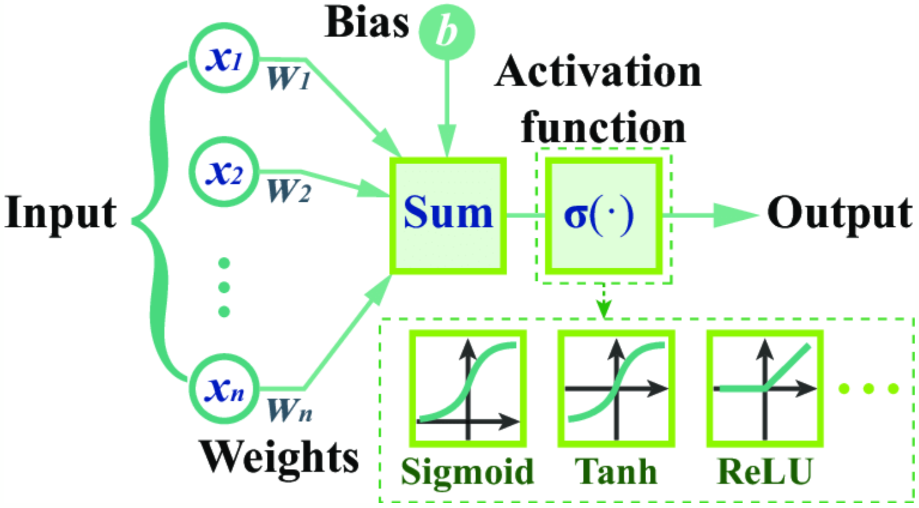


Non-linearity in Forward Propagation

During Forward Propagation:

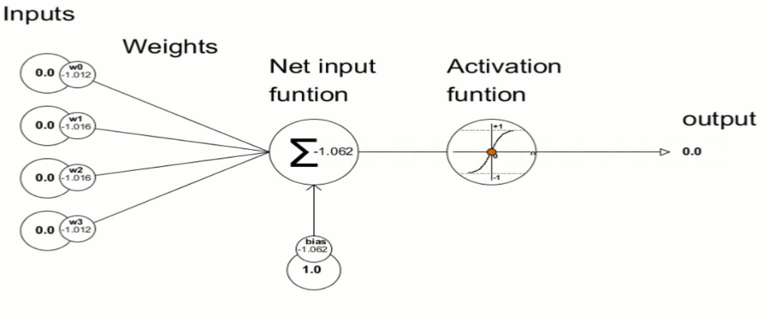
- **Pre-activation:** Calculates the weighted sum of inputs and previous hidden state.

- **Activation:** Introduces non-linearity based on the weighted sum using an activation function and bias.



Backpropagation

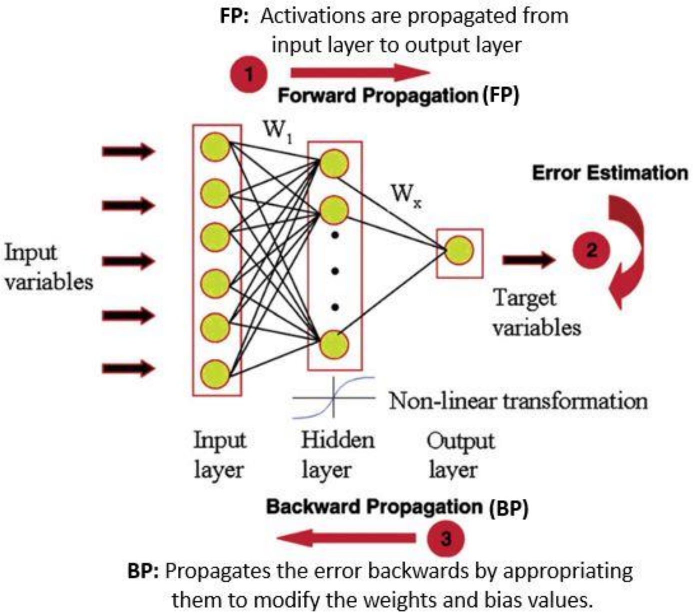
- Compute error between predicted and target outputs.
- Calculate gradients using backpropagation through time.
- Propagate gradients backward in time.
- Update parameters using optimization algorithms.
- Adjust parameters across epochs to minimize error.
- Address dependencies in long sequences.
- Use gradient clipping or learning rate scheduling.
- Trained RNN can generate text, complete sentences, or predict sequences.



Why we need backpropagation?

- Efficient and simple
- Few tunable parameters
- Flexibility
- Standard and effective
- General function learning

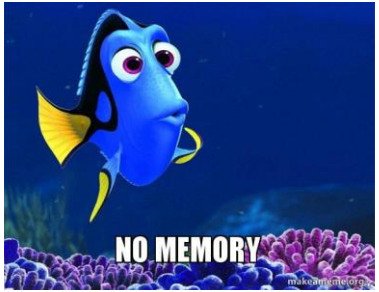
Workflow

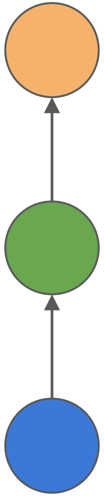

FP: Activations are propagated from input layer to output layer
1
Forward Propagation (FP)
W1
Wx
Error Estimation
2
Input variables
Target variables
Non-linear transformation
Input layer
Hidden layer
Output layer
Backward Propagation (BP)
3
BP: Propagates the error backwards by appropriating them to modify the weights and bias values.

Training & Optimization

01	Data Preparation	• Data collection • Data preprocessing • Sequence generation
02	Choice of Model Architecture	• Select RNN type • Define architecture • Activation function • Output layer
03	Training & Optimization	• Loss function • Optimization algorithm • Hyperparameter tuning • Training & validation • Evaluate & fine-tune

ANNs vs RNNs

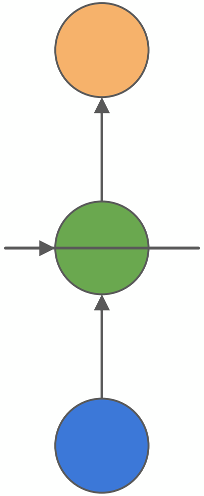


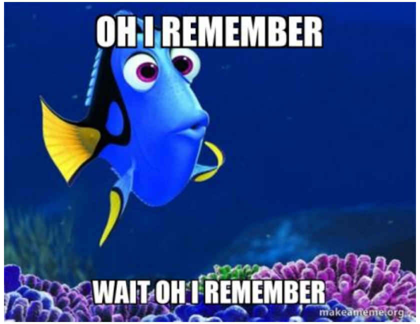


Output Layer

Hidden Layer

Input Layer





ANN Architecture

RNN Architecture

ANNs vs RNNs

ANNs	RNNs
Weights are different for each layer of the network	Weights are same across all the layers number of an RNN
A simple ANN doesn’t have any special method for sequential data; also here the number of inputs is fixed	RNNs are used when the data is sequential and the number of inputs is not predefined
The number of parameters are lower than RNN	The number of parameters are higher than ANN

Limitations of Traditional RNNs

Capturing Long-Term Dependencies Challenges

- RNNs is fine when dealing with short term dependencies.
  - Example: The longest river on Earth is **Nile**
- If the sequence is long enough he'll have a hard time carrying information from earlier time steps to later ones.
  - Example: The man who ate my pizza has **black hair**

- Backpropagation:

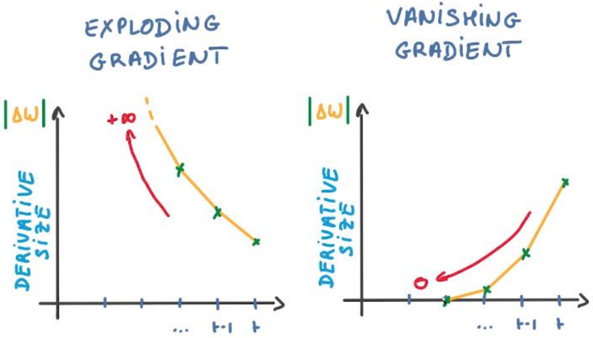
$$\partial E/\partial W = \partial E/\partial y3 * \partial y3/\partial h3 * \partial h3/\partial y2 * \partial y2/\partial h1$$

All the gradients would rush to zero exponentially fast due to the multiplication:

Vanishing Gradient

Exploding Gradients

- **Exploding gradients** are a problem when large error gradients accumulate and result in very large updates to neural network model weights during training.
- Gradients would rush to large values and eventually blow up and crash the mode.
- In this case, RNNs assign stupidly high importance to the weights without much reason.

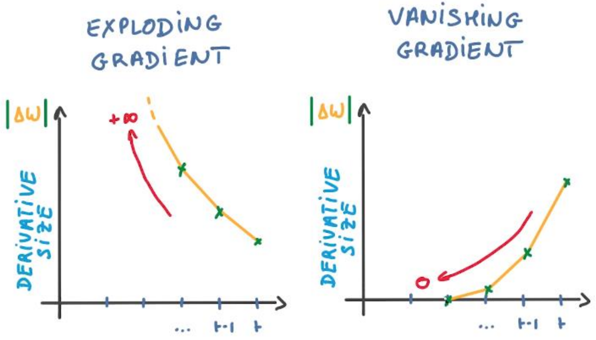


SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

أكاديمية سدايا
SDAIA Academy

The Long Short Term Memory LSTM

- RNNs can face issues:
  - vanishing gradients: weight changes that quickly became so small as to have no effect
  - exploding gradients: weight changes that became so large as to result in very large changes or even overflow
- LSTMs overcome this challenge by design.
- Comprised of layers of neurons
- Have recurrent connections so that the state from time steps is used as context for formulating an output.



Different Variants of RNNs

Well-Known Variants

LSTM

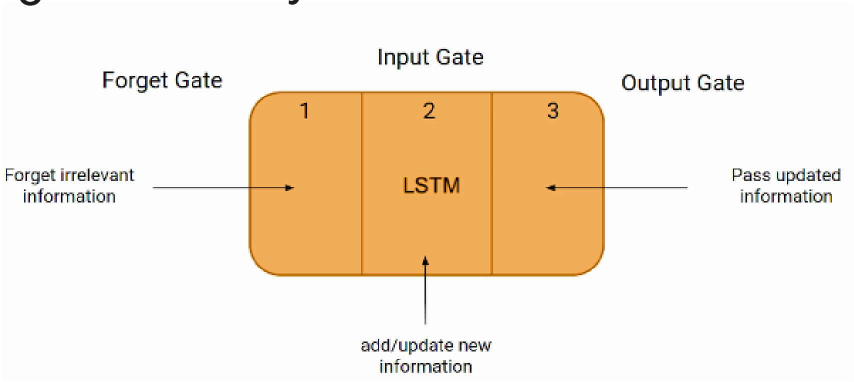
GRU

BRNN

Seq2Seq

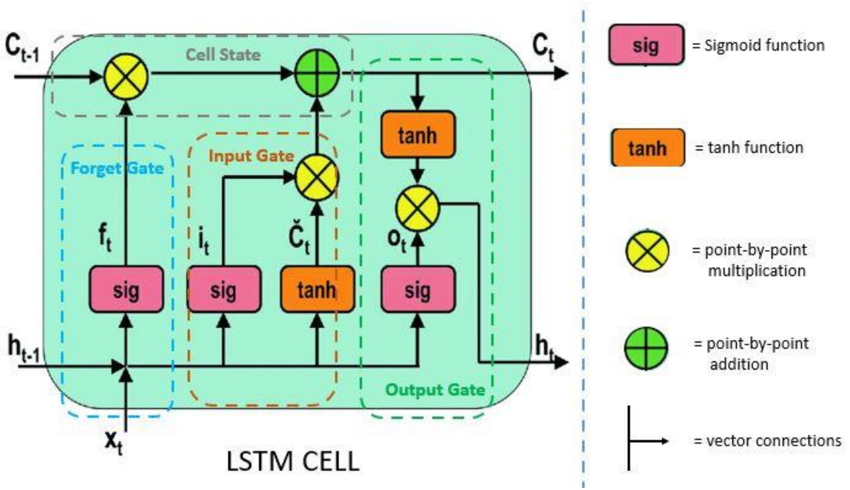
Intro to LSTM

- Key component: memory cell controlled by input, forget, and output gates
- Variant of RNN addressing vanishing/exploding gradients and capturing long-term dependencies
- Better modeling for memory and context tasks



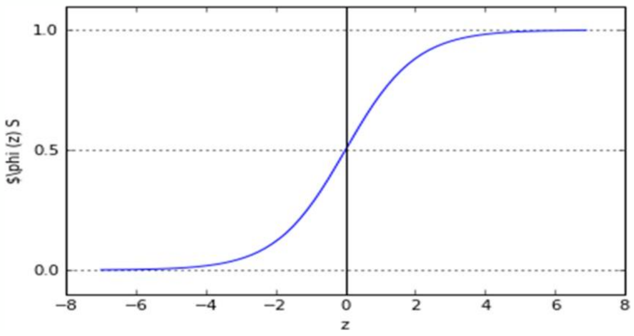
LSTM Architecture & Functioning

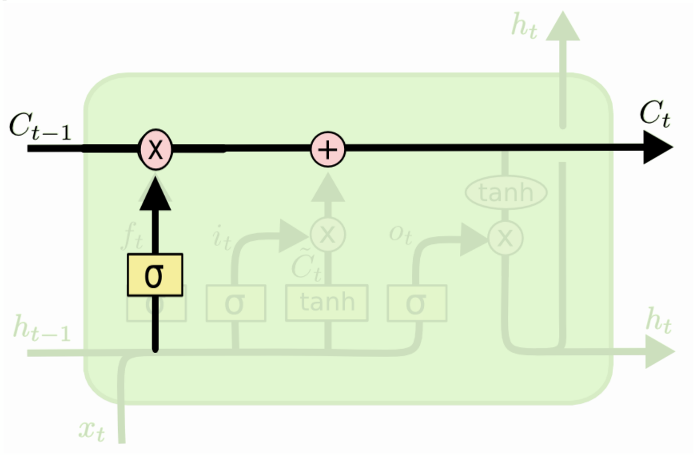
- Gates controlled by sigmoid functions to open/close based on input and hidden state
- Separate hidden state and memory cell for short-term and long-term information
- Effective in natural language processing, speech recognition, and time series analysis

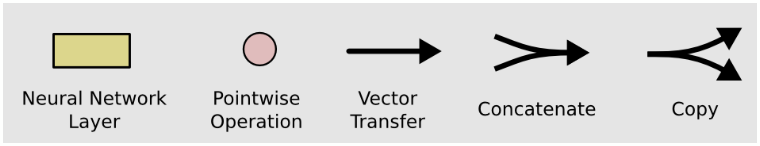


The Long Short Term Memory LSTM

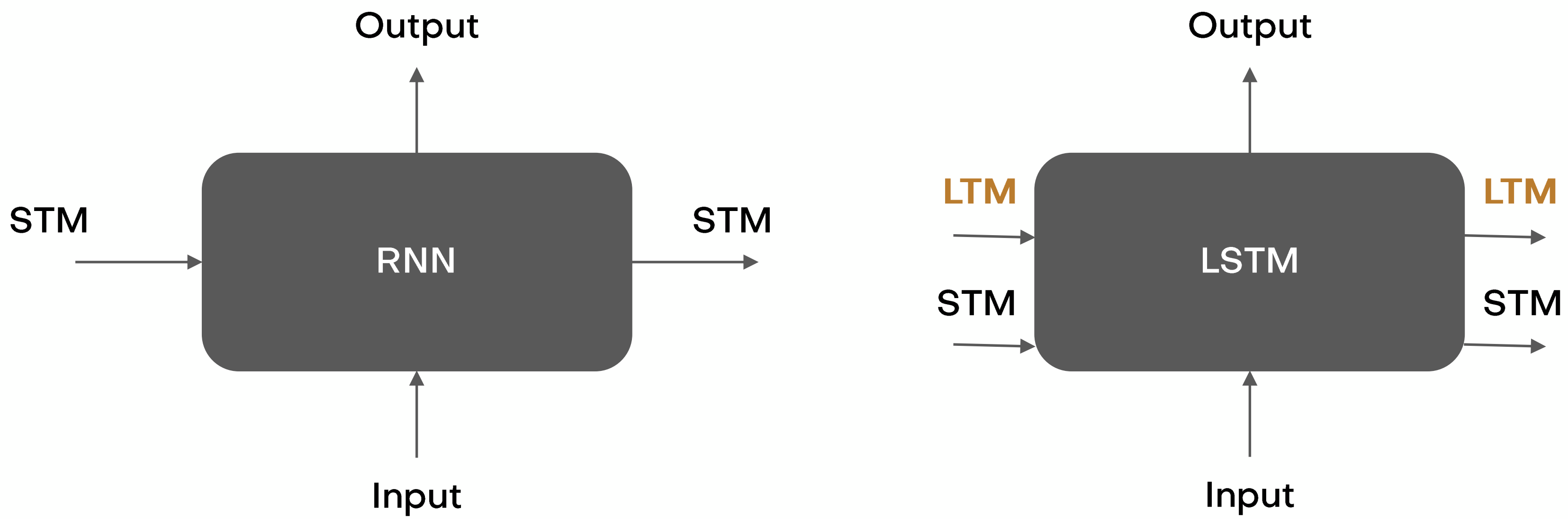
- Gate = Sigmoid Activation Function





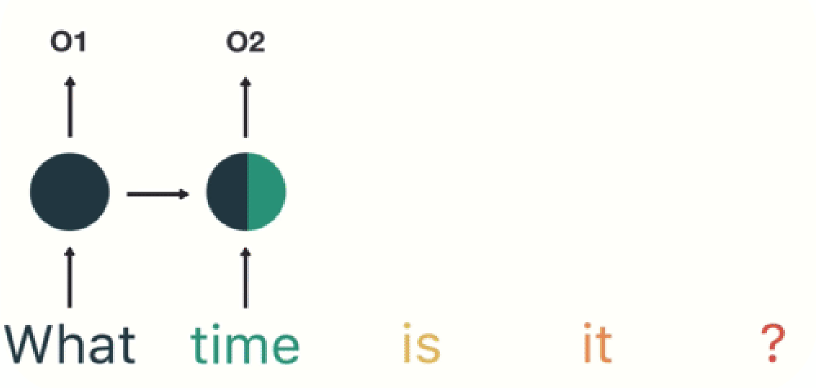


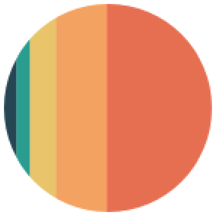
From RNN to LSTM



LSTM - A Better RNN

- Older inputs barley affects the output – also known as *Vanishing Gradient*
- Long Short-Term Memory (LSTM) Units resolve this issue
- Introduced a gating mechanism that can update the Cell State vector so that the context is now divided into STM and LTM and can be maintained across time

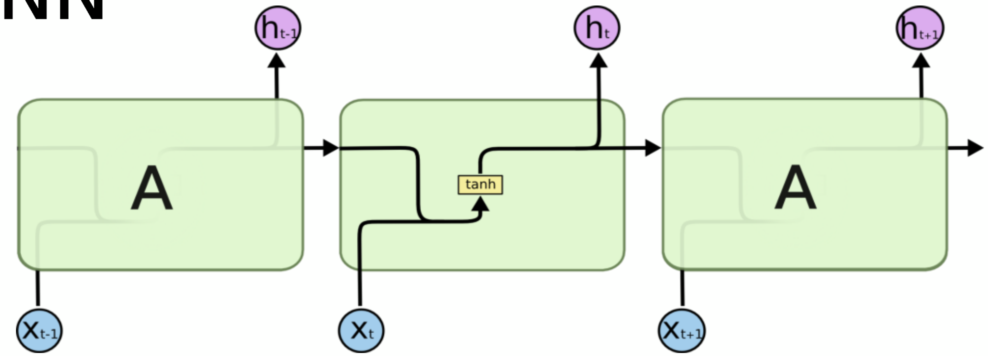




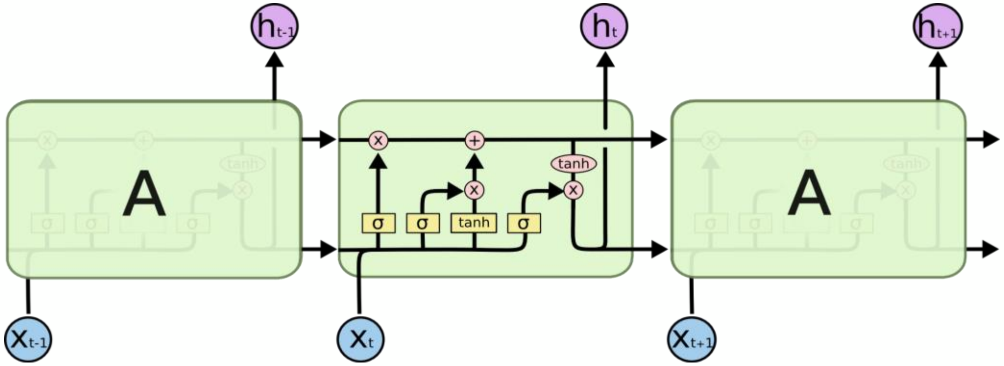
Final hidden state

LSTM - A Better RNN

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



The repeating module in an LSTM contains four interacting layers.



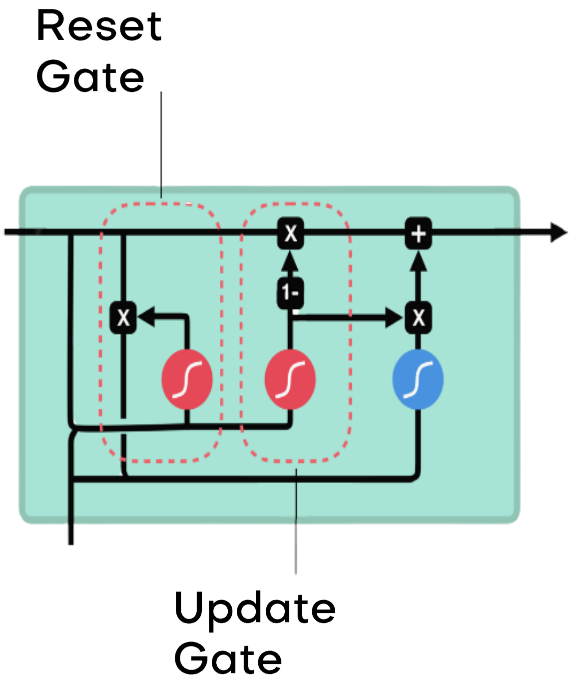
Intro to GRU

- Variant of RNNs addressing vanishing/exploding gradients and capturing complex dependencies with gating mechanisms

- Has two main gates: update and reset, controlling information updates and retention

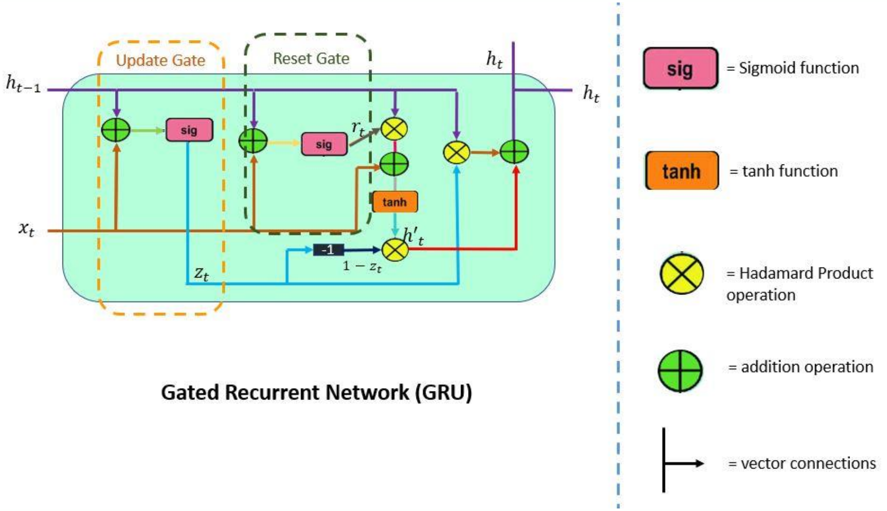
- Computationally efficient for limited resources

- Successful in NLP, speech recognition, and machine translation



GRU Architecture

- GRU combines previous hidden state and current input to compute a candidate activation.
- It uses the update gate to compute the current hidden state.



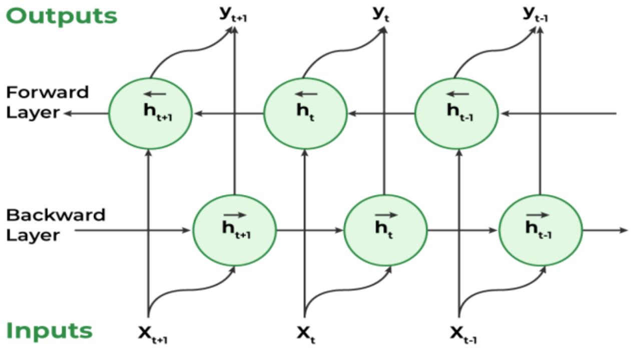
Gated Recurrent Network (GRU)

LSTM vs GRU

LSTM	GRU
Has 3 gates	Has 2 gates
More complex	Less complex
Preferred for larger dataset	Preferred for smaller dataset
Possesses internal memory	Doesn’t possess internal memory
Has an output gate	Doesn’t have an output gate

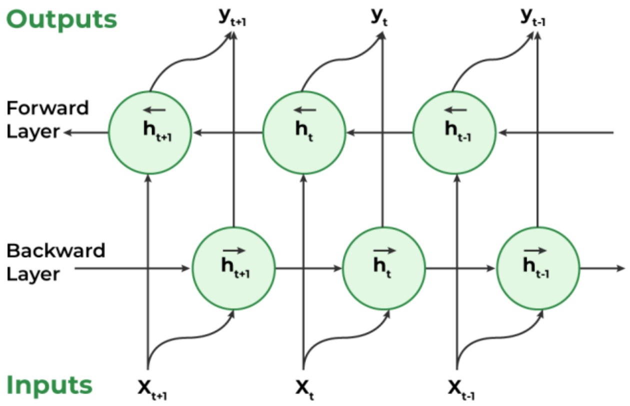
Bidirectional RNNs

- Bidirectional RNNs process sequential data in both forward and backward directions, capturing information from past and future context simultaneously
- Excel in tasks where contextual understanding in both directions is crucial, such as sentiment analysis and speech recognition



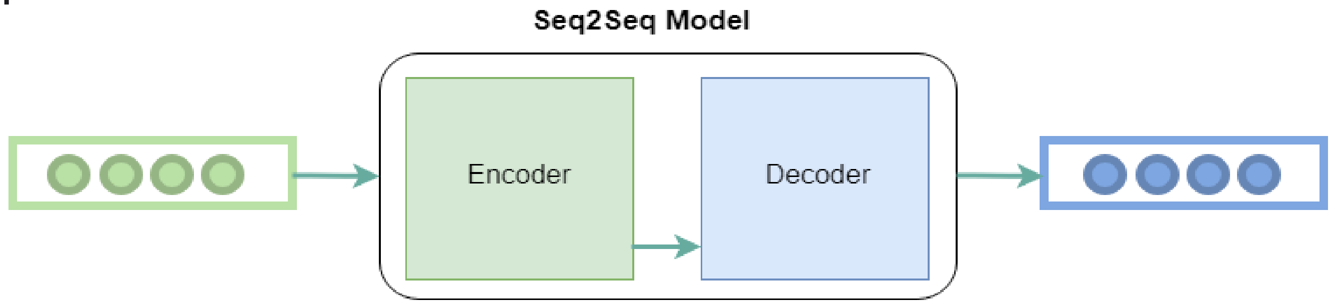
BRNN Architecture

- Input sequence presented in two passes: one in original order, and the other in reverse.
- Forward and backward hidden states computed independently, providing two sets of representations.
- Final hidden state combines information from both passes, incorporating context from past and future elements.



Seq2Seq

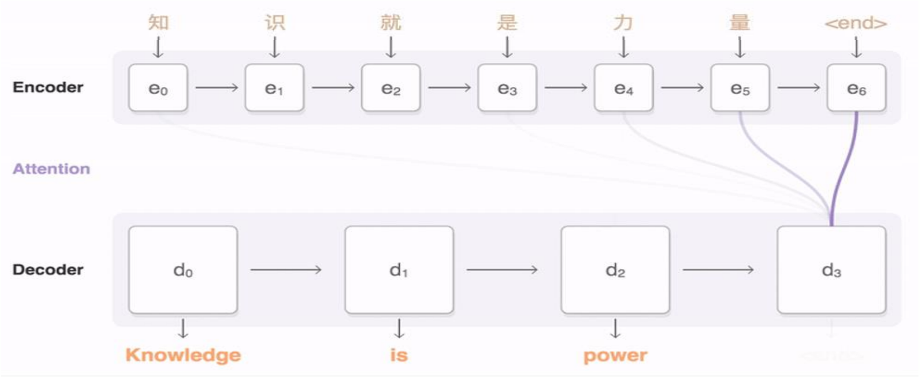
- Variant of RNNs for tasks like language generation, translation, and sequence prediction.
- Comprises encoder RNN and decoder RNN for input-to-output mapping.
- Successful in machine translation, text summarization, and chatbot responses.



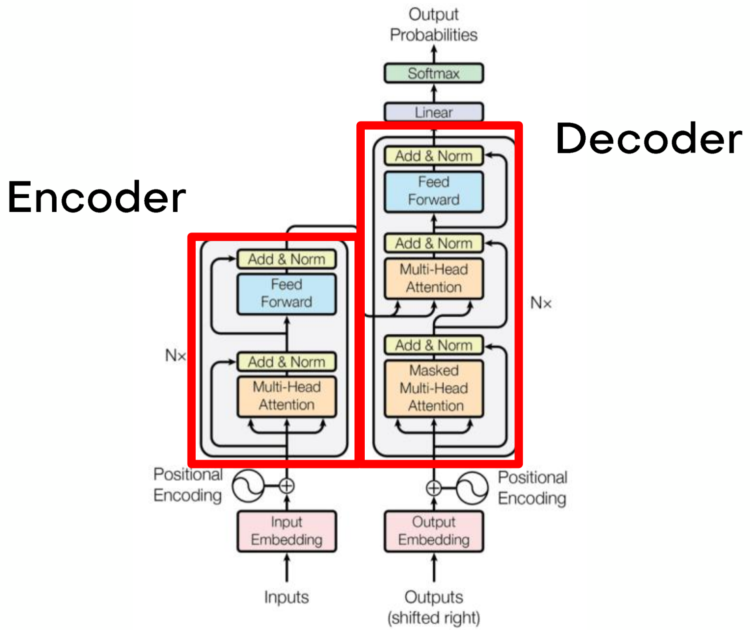
Alternative Solution to Traditional RNNs

The Transformer Architecture

- Transformer architecture revolutionized NLP and sequential data processing.
- Replaced RNNs with self-attention mechanisms for capturing long-range dependencies.
- Parallelizable and efficient, making it suitable for large-scale applications.
- Has become the backbone of models like BERT, GPT-3, and more.

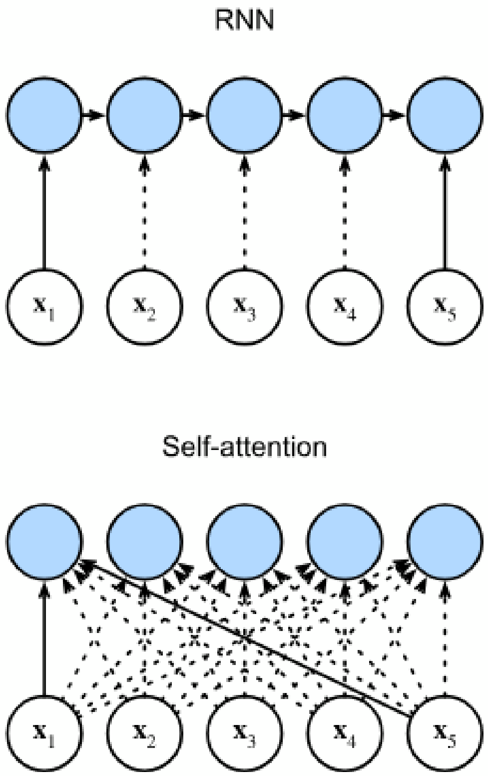


Transformer Main Components



Self-Attention Mechanism

- Self-attention is a key component of the Transformer architecture.
- It allows the model to focus on different parts of the input sequence to learn meaningful representations.
- Self-attention enables a better understanding of context and dependencies among words in the sequence.
- This mechanism plays a crucial role in the success of Transformers in various natural language processing tasks.



Hands-on: Text Generation using GPT-2

شكراً لكم

Thank you

SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority