

Deep Learning & Neural Networks

Dr. Shahira Ahmed

DS710

Lecture 10





Deep Learning & Neural Networks DS710

Lecture 10
Recurrent Neural Networks
RNN



LEARNING OUTCOMES

- ◉ Understand the basic concepts and architecture of recurrent neural networks
- ◉ Identify the different types of RNNs and their applications
- ◉ Understanding the flow of information in an RNN



LECTURE OUTLINE

- ◉ Introduction to Recurrent Neural Networks
- ◉ Architecture of Recurrent Neural Networks
- ◉ Applications of Recurrent Neural Networks
- ◉ Types of RNNs: Simple RNN, LSTM, and GRU
- ◉ Lab 1
- ◉ Lab2

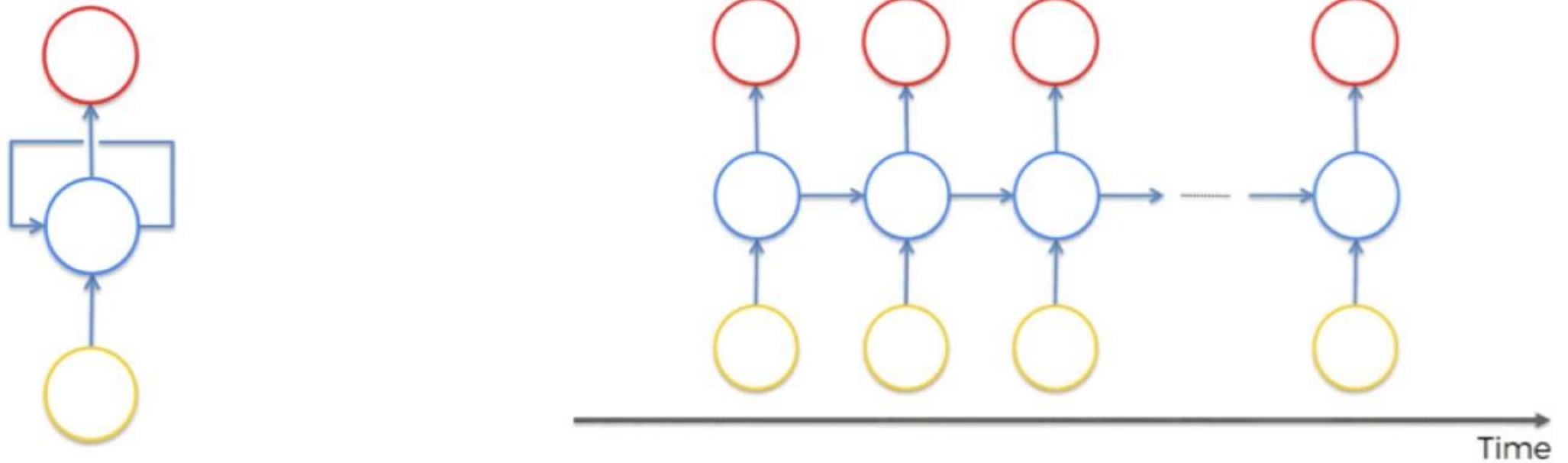
- A sequence refers to a collection of data points that are ordered based on their **position** or **time**.
- Each element in the sequence has a **specific position** or **index** that determines its order relative to other elements in the sequence.

- **Numeric Sequences:** A sequence of numbers, such as [1, 3, 5, 7, 9], sensor reading.
- **Text Sequences:** A sequence of characters or words, such as ["Hello", "world", "!"]. Text sequences are often used in natural language processing tasks, such as language modeling, sentiment analysis, or machine translation.
- **Time Series:** A sequence of observations recorded over time, such as daily temperature readings or stock market prices. Time series analysis involves analyzing and forecasting patterns in these sequences.

- **DNA Sequences:** A sequence of nucleotides (**A, C, G, T**) representing genetic information. DNA sequences are used in genomics research to study genes, mutations, and evolutionary relationships.
- **Music Sequences:** A sequence of musical notes or chords representing a piece of music. Music sequences can be analyzed and used for tasks such as music generation or recommendation.
- **Video Frames:** A sequence of images or frames captured from a video. Video sequences can be processed for tasks like object recognition, action recognition, or video summarization

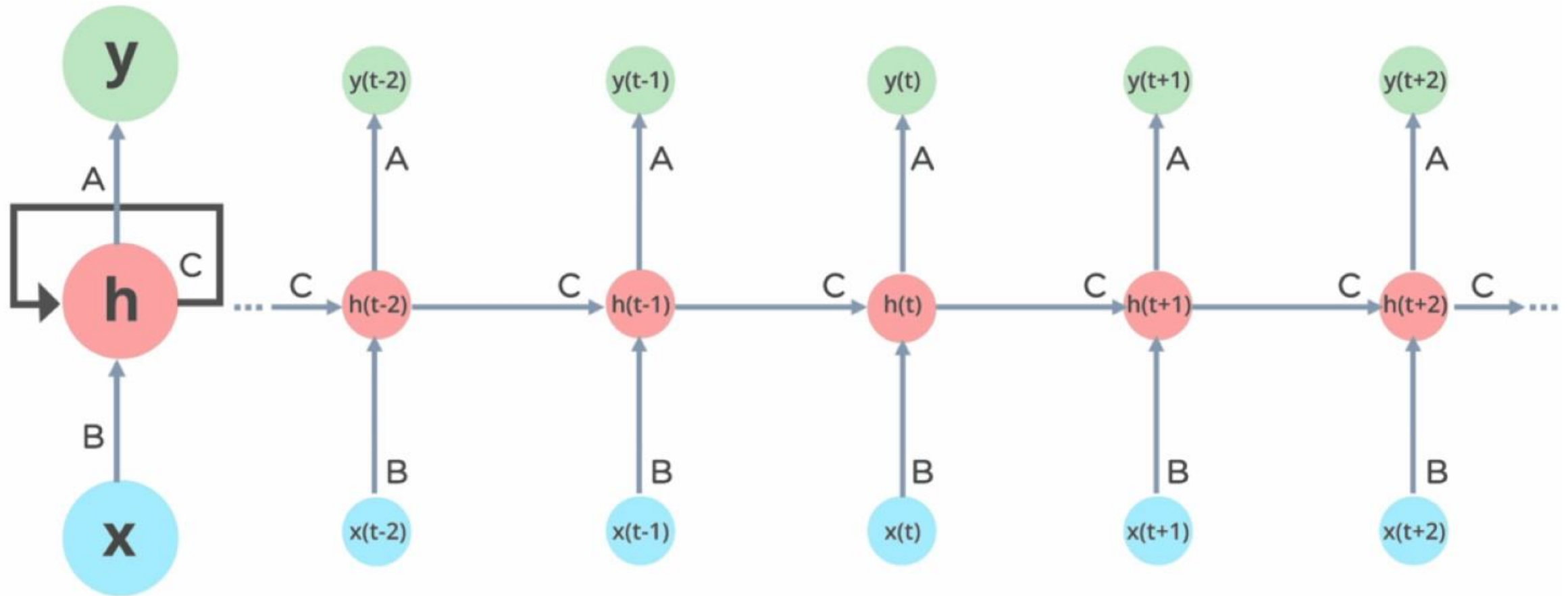
- Recurrent Neural Networks (RNNs) are a type of neural network architecture specifically designed to model and **process sequential data**.
- Unlike traditional feedforward neural networks, RNNs have connections that allow information to flow in **cycles**, enabling them to capture dependencies and patterns in sequential data.
- RNN 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.

Recurrent Neural Networks (RNNs)



- [The Ultimate Guide to Recurrent Neural Networks \(RNN\) - Blogs SuperDataScience - Machine Learning | AI | Data Science Career | Analytics | Success](#)

How Does Recurrent Neural Networks Work?



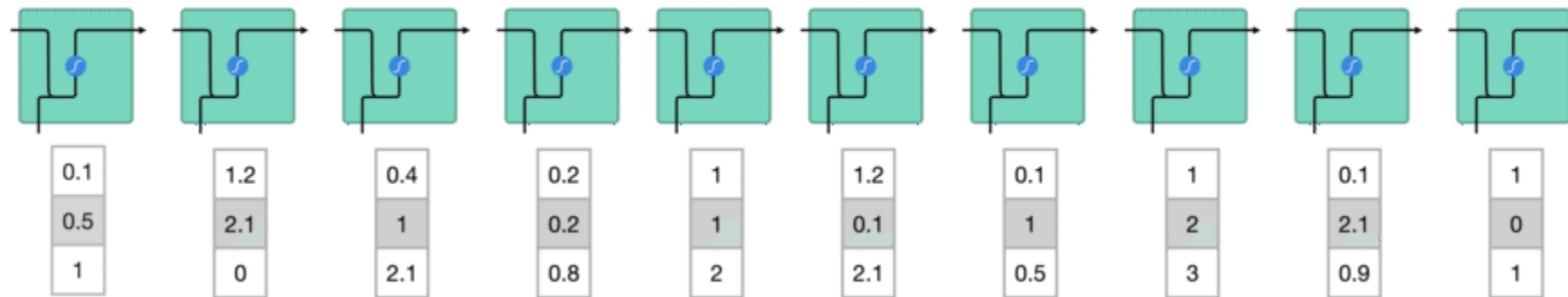
<https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>

- Recurrent Neural networks, the information cycles through a loop to the middle hidden layer.
- Input and Hidden State:
 - At each step of the sequence, an RNN receives an input vector or data point. For example, in natural language processing, the input vector can represent a word or a character.
 - In addition to the input vector, the RNN also has a hidden state vector, which represents the network's memory and captures information from previous steps. Initially, the hidden state is initialized to a fixed value or learned from data

- Information Flow and Recurrent Connections:
 - The input vector and the hidden state are combined and passed through a set of learnable weights or parameters.
 - The output of this computation is used to update the hidden state for the current step, which becomes the contextual information for the next step.
 - Importantly, the hidden state is not only influenced by the current input but also influenced by the hidden state from the previous step. This feedback loop creates recurrent connections that allow information to flow across different steps in the sequence.

How Does Recurrent Neural Networks Work?

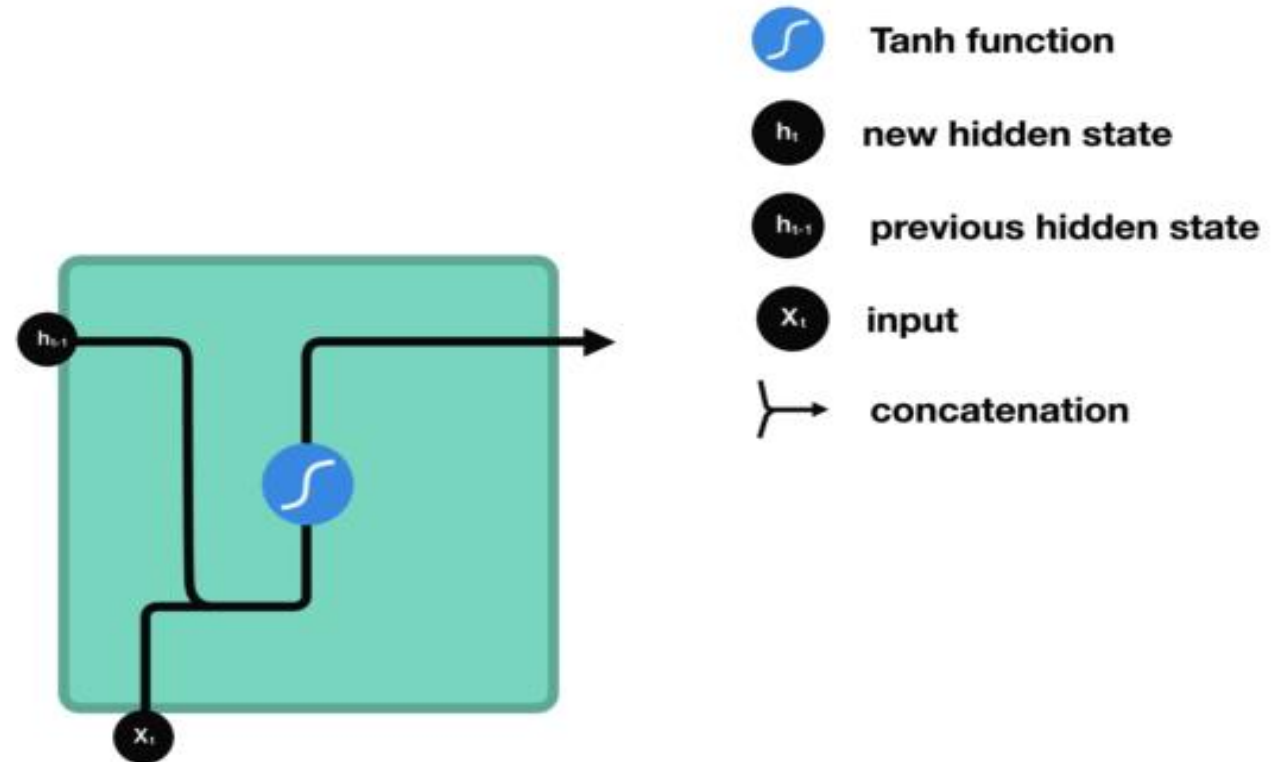
(1)- words get transformed into machine-readable vectors. Then the RNN processes the sequence of vectors one by one.



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

How Does Recurrent Neural Networks Work?

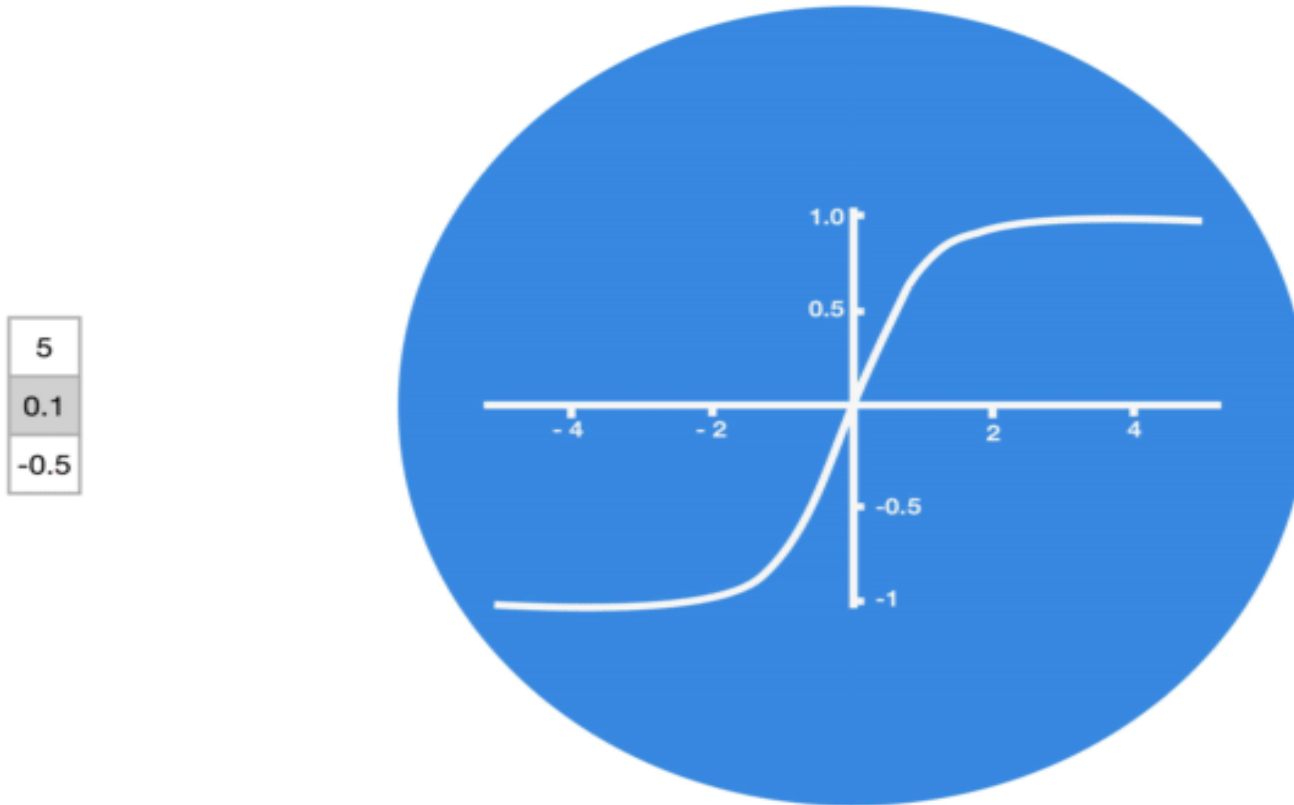
- RNN Cell :the input and previous hidden state are combined to form a



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

How Does Recurrent Neural Networks Work?

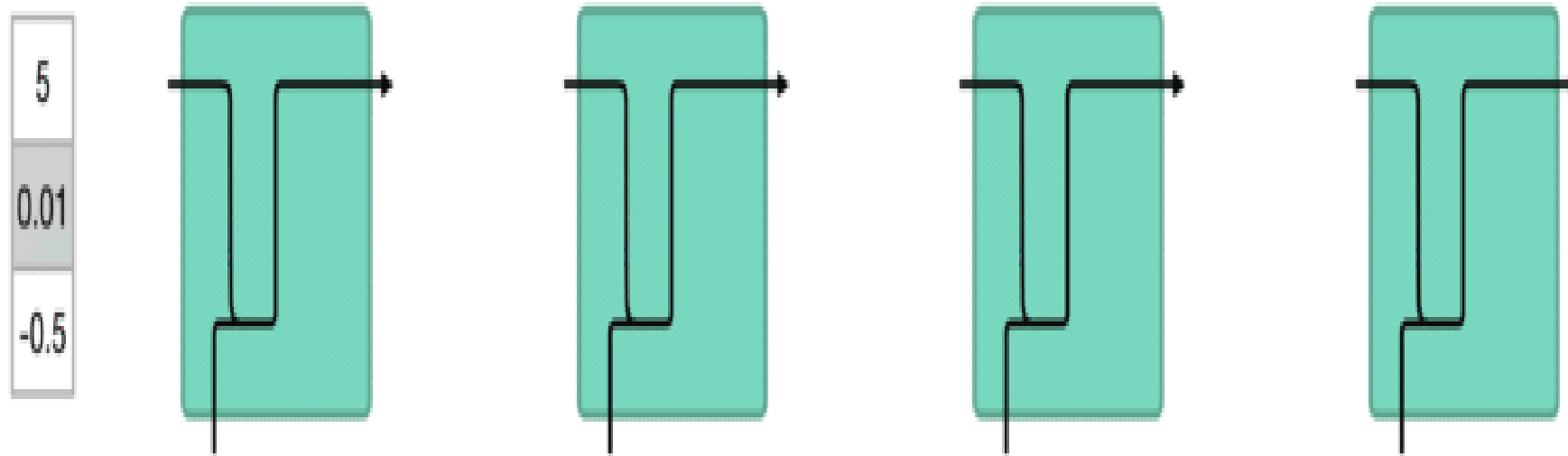
- Tanh squishes values to be between -1 and 1



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

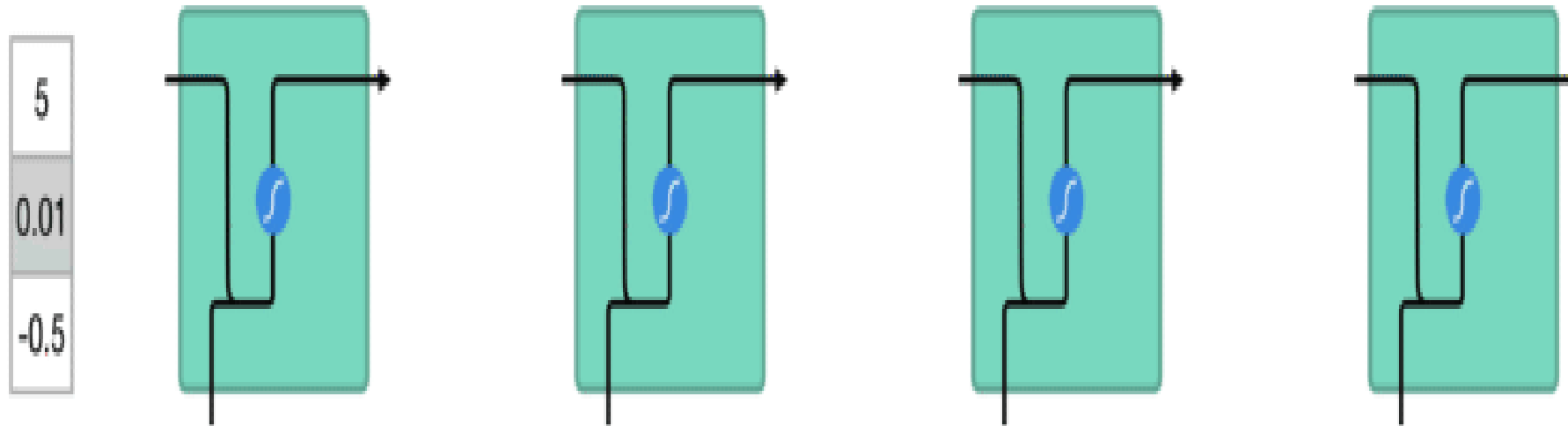
How Does Recurrent Neural Networks Work?

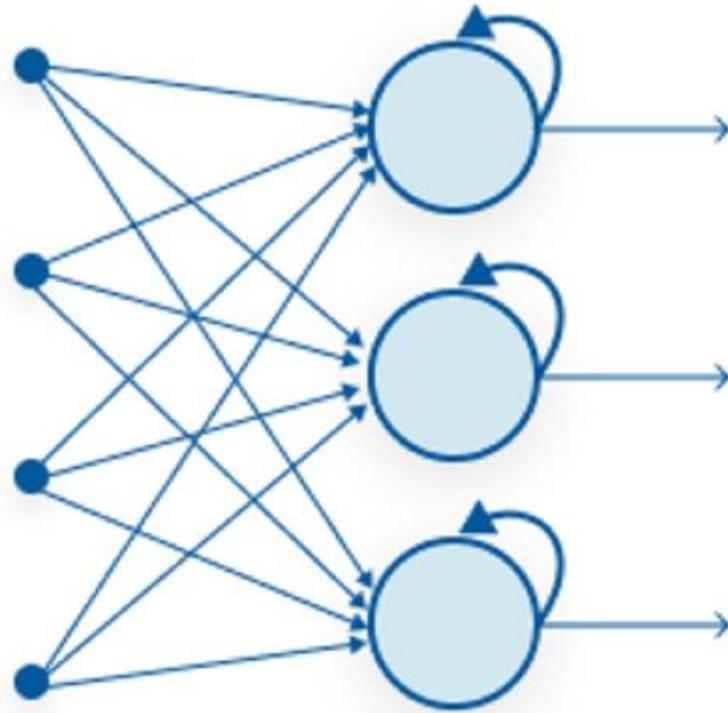
- When vectors are flowing through a neural network, it undergoes many transformations due to various math operations.



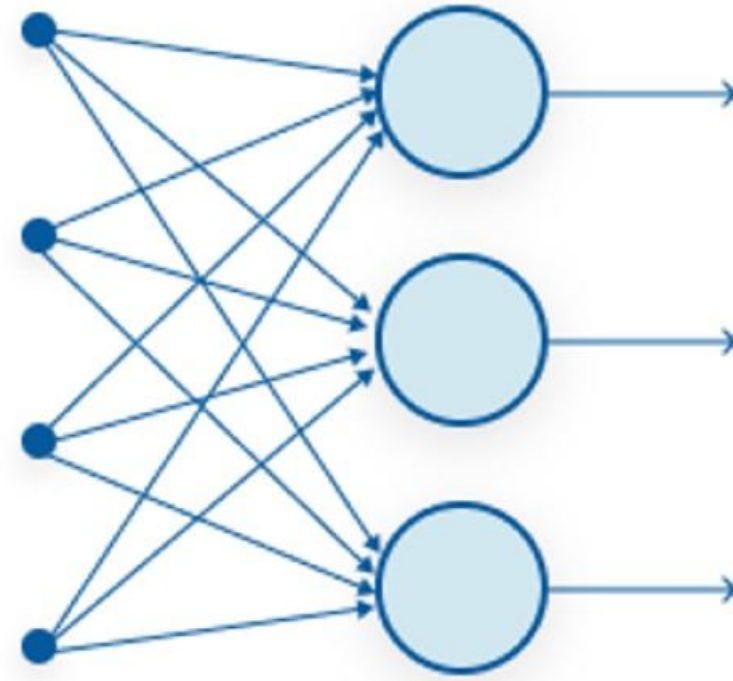
How Does Recurrent Neural Networks Work?

- A tanh function ensures that the values stay between -1 and 1, thus regulating the output of the neural network. You can see how the same values from above remain between the boundaries allowed by the tanh function.





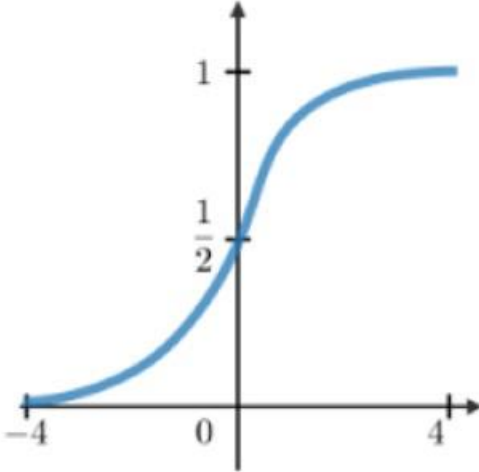
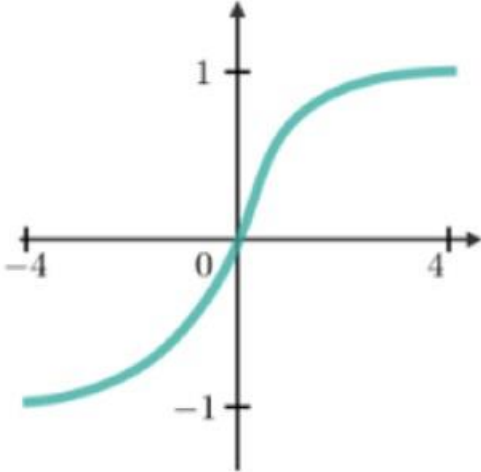
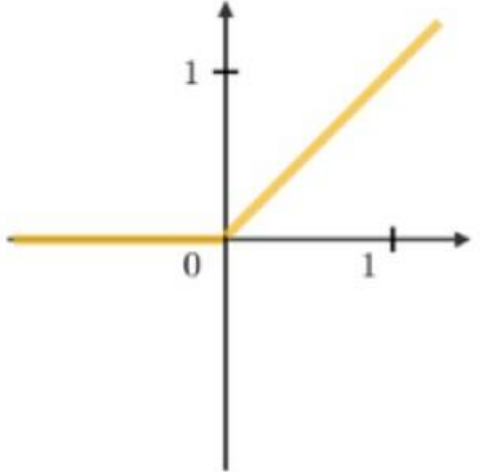
Recurrent Neural Network



Feed-Forward Neural Network

Source: Uditvani.com

Common Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

- Feed-forward neural networks are **poor predictions** of **sequences** because they have **no memory** of the information they receive.
- Feed-forward neural networks simply analyses the current input, a feed-forward network has no idea of **temporal order**. Apart from its training, it has no memory of what transpired in the past.

- The information is in an RNN cycle via a **loop**. Before making a judgment, it evaluates the current input as well as what it has learned from past inputs.

- RNNs are naturally suited for processing sequential data with **varying lengths**. They inherently have a memory component that allows them to **maintain** information **from previous time steps**.
- RNNs can handle **variable-length** input sequences by processing them one time step at a time. The RNN's hidden state is updated at each time step, allowing it to capture dependencies and patterns in the sequence.

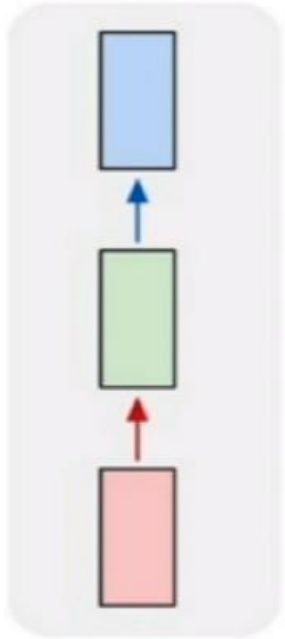
- Padding is adding zeros or a special token to the end of shorter sequences to make them have the same length as the longest sequence in a batch.
- Masking is telling the RNN to ignore the padded values and only process the actual inputs¹. You can use a mask tensor that has ones where the inputs are valid and zeros where they are padded.
- Single-sample batches are batches that contain only one sequence, so there is no need for padding or masking. However, this can be inefficient and slows

- FFNs, on the other hand, are typically designed to handle fixed-length input vectors. They do not have an inherent memory component like RNNs, and their layers are fully connected, meaning each input element is treated independently.
- you can also use techniques like **padding or truncation** to make the input vectors uniform in length

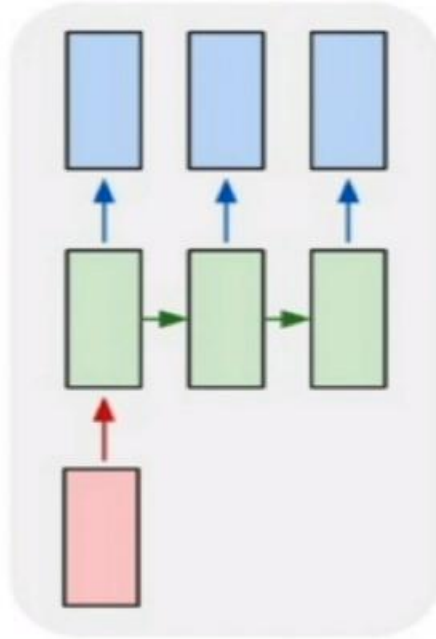
- RNN were created because there were a few issues in the feed-forward neural network:
 - Cannot handle sequential data
 - Cannot handle variable length input
 - Considers only the current input
 - Cannot memorize previous inputs

Main Types of RNN Architectures

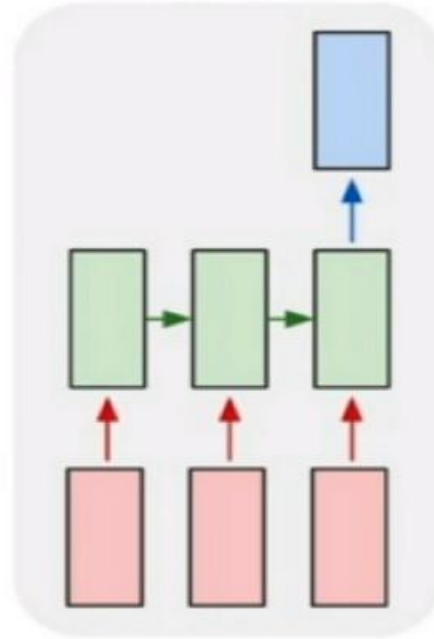
one to one



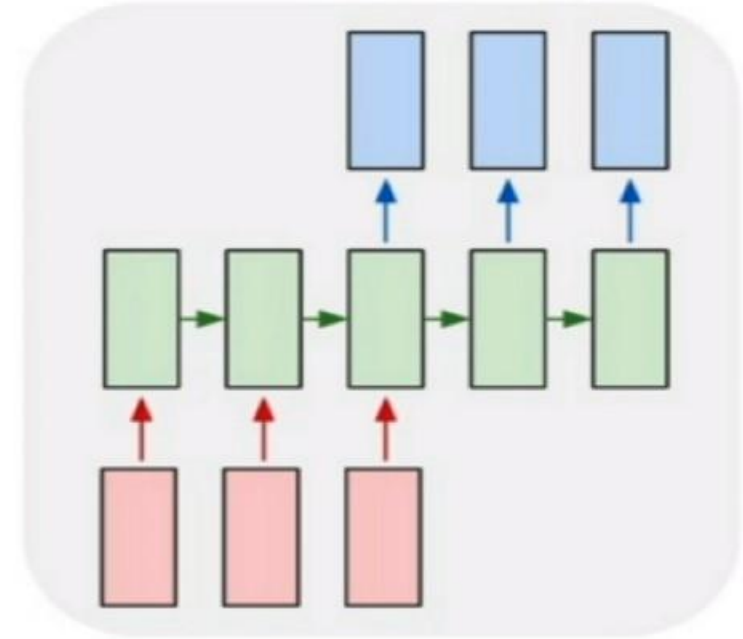
one to many



many to one



many to many



- **One-to-One (Feedforward)**: This is not technically an RNN architecture but a traditional feedforward neural network where there is no recurrence. It takes fixed-size input and produces a fixed-size output without considering any temporal dependencies.
- **One-to-Many**: In this architecture, the RNN takes a single input and generates a sequence of outputs. For example, generating a sentence given an image caption.
- **Many-to-One**: In this architecture, the RNN takes a sequence of inputs and produces a single output. For example, sentiment analysis where the input is a sentence and the output is the sentiment of that sentence.
- **Many-to-Many (Sequence-to-Sequence)**: This architecture takes a sequence of inputs and produces a sequence of outputs. Many-to-many This model is ideal for machine translation like the one we see on Google translate. The input could be an English sentence which has variable length and the output will be the same sentence in a different language which also has variable length

- The Problem: **Short-term Memory**
 - Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.
 - During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- While training a neural network, if the slope tends to **grow exponentially** instead of **decaying**, this is called an **Exploding Gradient**. This problem arises when large error gradients accumulate, resulting in very large updates to the neural network model weights during the training process.

- One popular variation is the Long Short-Term Memory (**LSTM**) network. LSTM networks introduce memory cells and gating mechanisms that allow for better preservation of information over longer sequences.
- Another variation is the Gated Recurrent Unit (**GRU**), which simplifies the LSTM architecture by combining the forget and input gates into a single update gate. GRUs have similar capabilities to LSTMs but with a slightly more streamlined structure.

- Recurrent Neural Networks (RNNs) have several advantages over other types of neural networks, including:
 - Ability To Handle Variable-Length Sequences : RNNs are designed to handle input sequences of variable length, which makes them well-suited for tasks such as speech recognition, natural language processing, and time series analysis.
 - Memory Of Past Inputs : RNNs have a memory of past inputs, which allows them to capture information about the context of the input sequence. This makes them useful for tasks such as language modeling, where the meaning of a word depends on the context in which it appears.
 - Parameter Sharing : RNNs share the same set of parameters across all time steps, which reduces the number of parameters that need to be learned and can lead to better generalization.

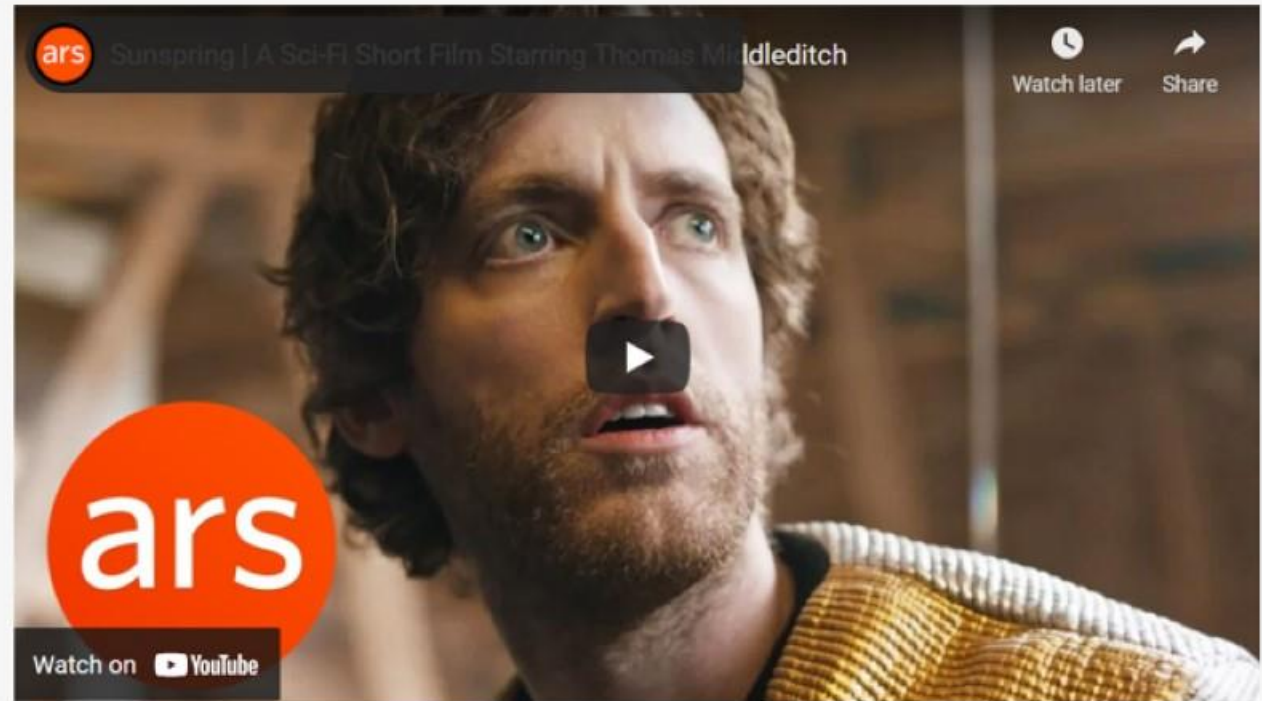
- **Vanishing/Exploding Gradient Problem:** RNNs suffer from the vanishing or exploding gradient problem, which occurs when gradients either become too small or too large during backpropagation. This makes it difficult for the network to learn long-term dependencies or propagate information effectively over long sequences.
- **Difficulty in Capturing Long-Term Dependencies:** Standard RNN architectures have difficulty capturing long-term dependencies in sequential data. As the time lag between relevant information increases, RNNs tend to forget or lose track of important information from earlier time steps.
- **Lack of Parallelism:** RNNs process sequential data one time step at a time, which makes them inherently sequential in nature. This limits their ability to take advantage of parallel computing, resulting in slower training and inference times compared to other neural network architectures.

- **Memory Limitations:** RNNs have a fixed-size hidden state that is shared across all time steps. This fixed memory capacity can be a limitation when dealing with long sequences or complex tasks that require capturing extensive context or information.
- **Difficulty with Variable-Length Inputs:** RNNs typically require fixed-length input sequences. Dealing with variable-length inputs, such as sentences of different lengths in natural language processing tasks, requires additional pre-processing or padding techniques, which can be inefficient.

- **Computational Complexity:** RNNs can be computationally expensive to train and require substantial computational resources due to their recurrent nature and sequential processing. Training large-scale RNN models on long sequences can be time-consuming and resource-intensive.

Applications of RNNs

- Sunspring (movie, 2016) •
- Directed By Oscar Sharp •
- Written Benjamin



- [Sunspring - The short film written by an AI robot called 'Benjamin'. – Acumass](#)
- <https://youtu.be/LY7x2lhqjmc>

- The following are some of the applications of RNNs:
 - **Natural Language Processing (NLP):** RNNs are widely used in NLP tasks such as **language modeling, machine translation, sentiment analysis, named entity recognition, and text generation**. They can effectively capture the sequential nature of language and model dependencies between words or characters in a sentence.
 - **Speech Recognition:** RNNs are used in automatic speech recognition systems to **convert spoken language into written text**. They can model the temporal dependencies in speech signals.
 - **Time Series Analysis:** RNNs are well-suited for time series analysis tasks such as **forecasting, anomaly detection, and signal processing**. They can capture temporal dependencies and patterns in sequential data, making them effective for predicting future values or detecting abnormal behavior.

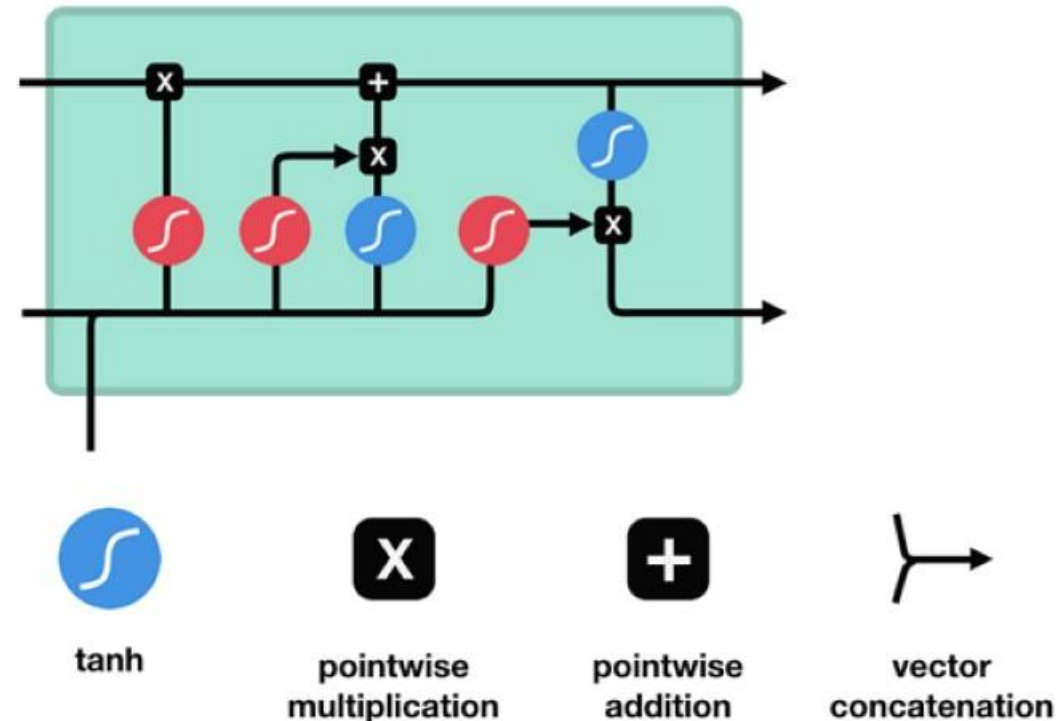
- The following are some of the applications of RNNs:
 - **Image Captioning:** RNNs combined with Convolutional Neural Networks (CNNs) are used for generating captions for images. CNNs extract visual features from images, which are then fed into an RNN to generate a textual description of the image.
 - **Music Generation:** RNNs can be used to generate music by learning from patterns in a sequence of musical notes or audio samples. They can capture musical structure and create new compositions based on the learned patterns.

- The following are some of the applications of RNNs:
 - **DNA Sequence Analysis:** RNNs can be useful for analyzing DNA sequences and identifying patterns or motifs within the sequences.
 - **Health Monitoring and Disease Detection:** RNNs can be used for health monitoring and disease detection by analyzing time series data from wearable devices or medical sensors. They can identify patterns or anomalies in physiological signals and assist in diagnosing diseases or monitoring patient health.
 - **Gesture Recognition:** RNNs can analyze sequential data from sensors or motion capture devices to recognize and interpret human gestures. This has applications in fields such as sign language recognition, human-computer interaction, and virtual reality.

- RNN's uses a lot less computational resources than its evolved variants, LSTM's and GRU's.
- **Long Short-Term Memory (LSTM):** LSTM is an extension of the traditional RNN architecture that introduces a memory cell, which allows for better handling of long-term dependencies. LSTMs use gates to control the flow of information into and out of the memory cell, enabling them to selectively retain or forget information over long sequences.
- **Gated Recurrent Unit (GRU):** GRU is another variant of the traditional RNN architecture that addresses the vanishing gradient problem. GRUs have a similar gating mechanism as LSTMs but with a simplified structure. They combine the forget and input gates into a single "update gate" and merge the hidden state and memory cell into a single "hidden state."

Long Short-Term Memory (LSTM)

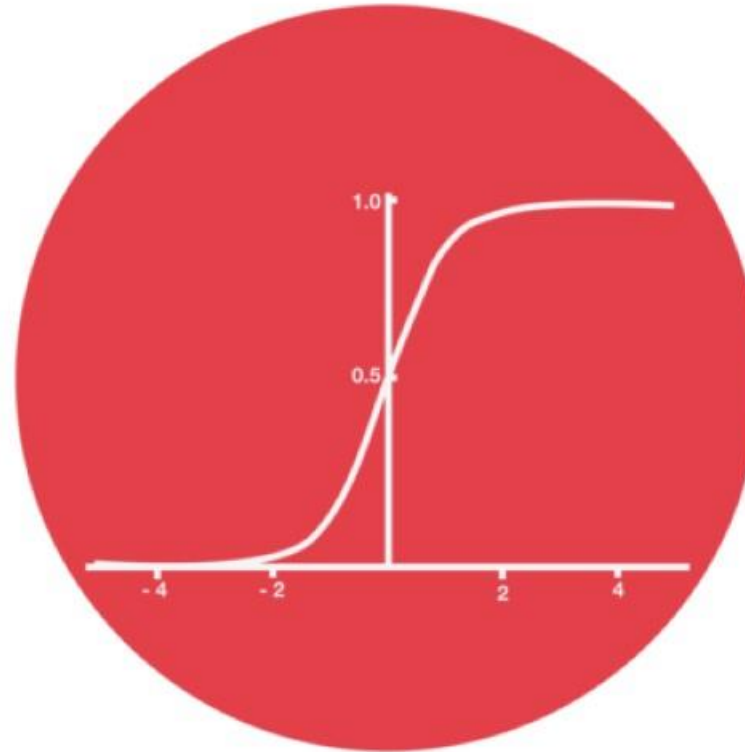
- An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward.
- The differences are the operations within the LSTM's cells.



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- **Sigmoid**

5
0.1
-0.5

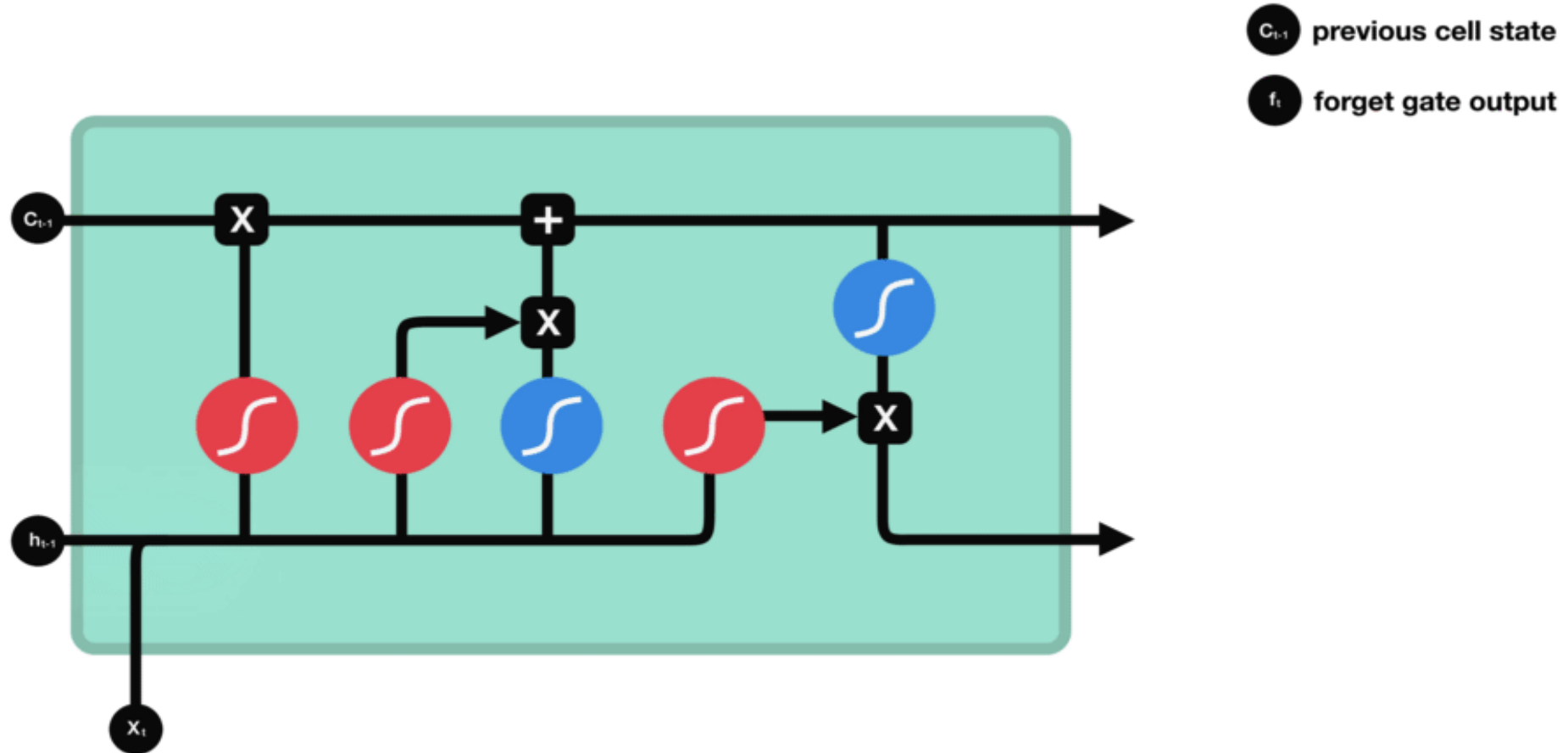


<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- **Sigmoid**
- A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1.
- That is helpful to update or **forget** data because any number getting **multiplied by 0** is 0, causing values to disappear or be “forgotten.”
- Any number **multiplied by 1** is the same value therefore that value stay's the same or is “**kept**.” The network can learn which data is not important therefore can be forgotten or which data is important to keep.

- **Forget gate**
- This gate **decides what information should be thrown away or kept.** Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. **The closer to 0 means to forget, and the closer to 1 means to keep.**

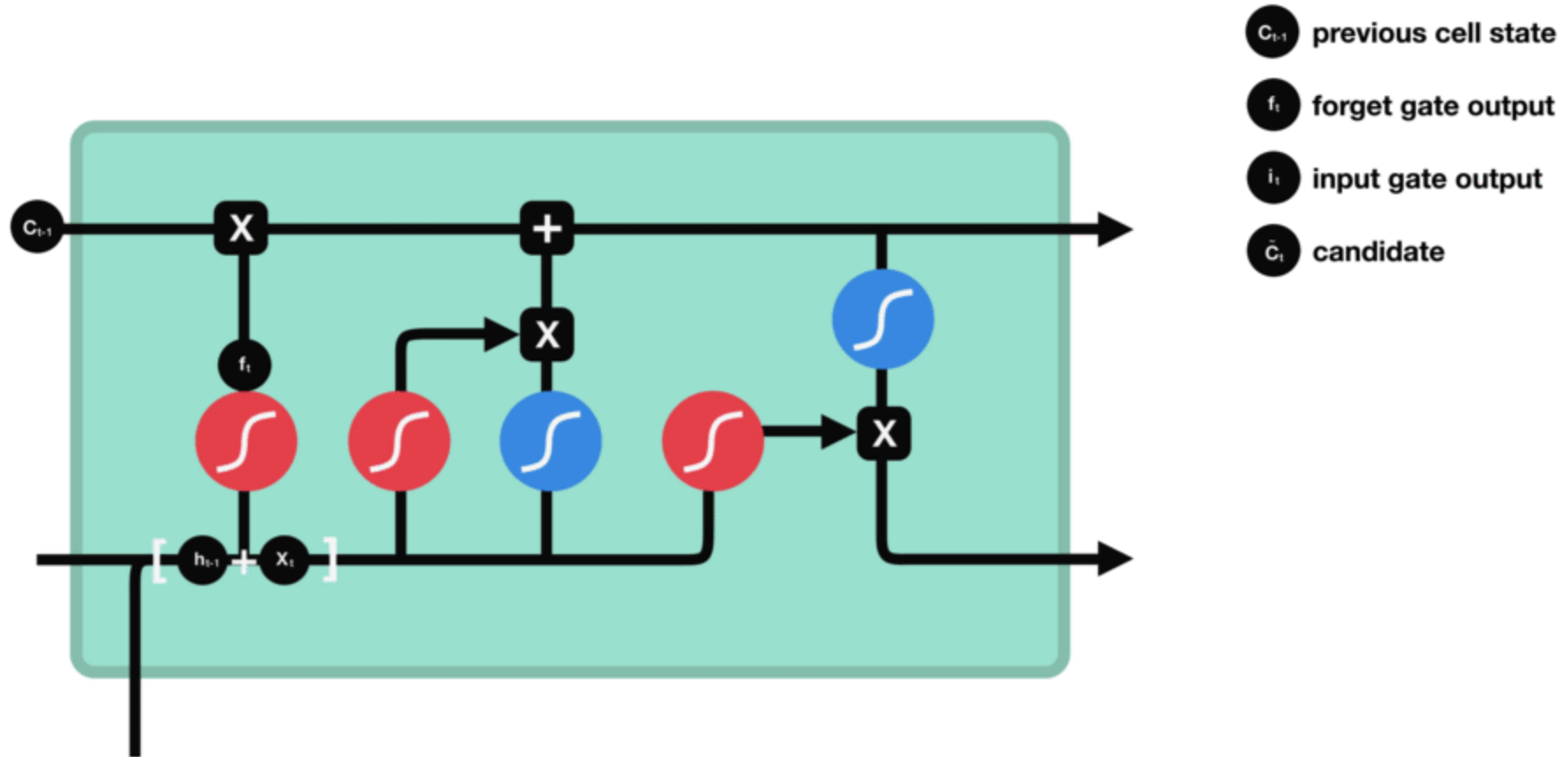
Long Short-Term Memory (LSTM)



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- **Input Gate**
- we pass the **previous hidden state and current input** into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important.
- You also **pass the hidden state and current input into the tanh function** to squish values between -1 and 1 to help regulate the network.
- Then you **multiply** the **tanh output with the sigmoid output**. The sigmoid output will decide which information is important to keep from the tanh output.

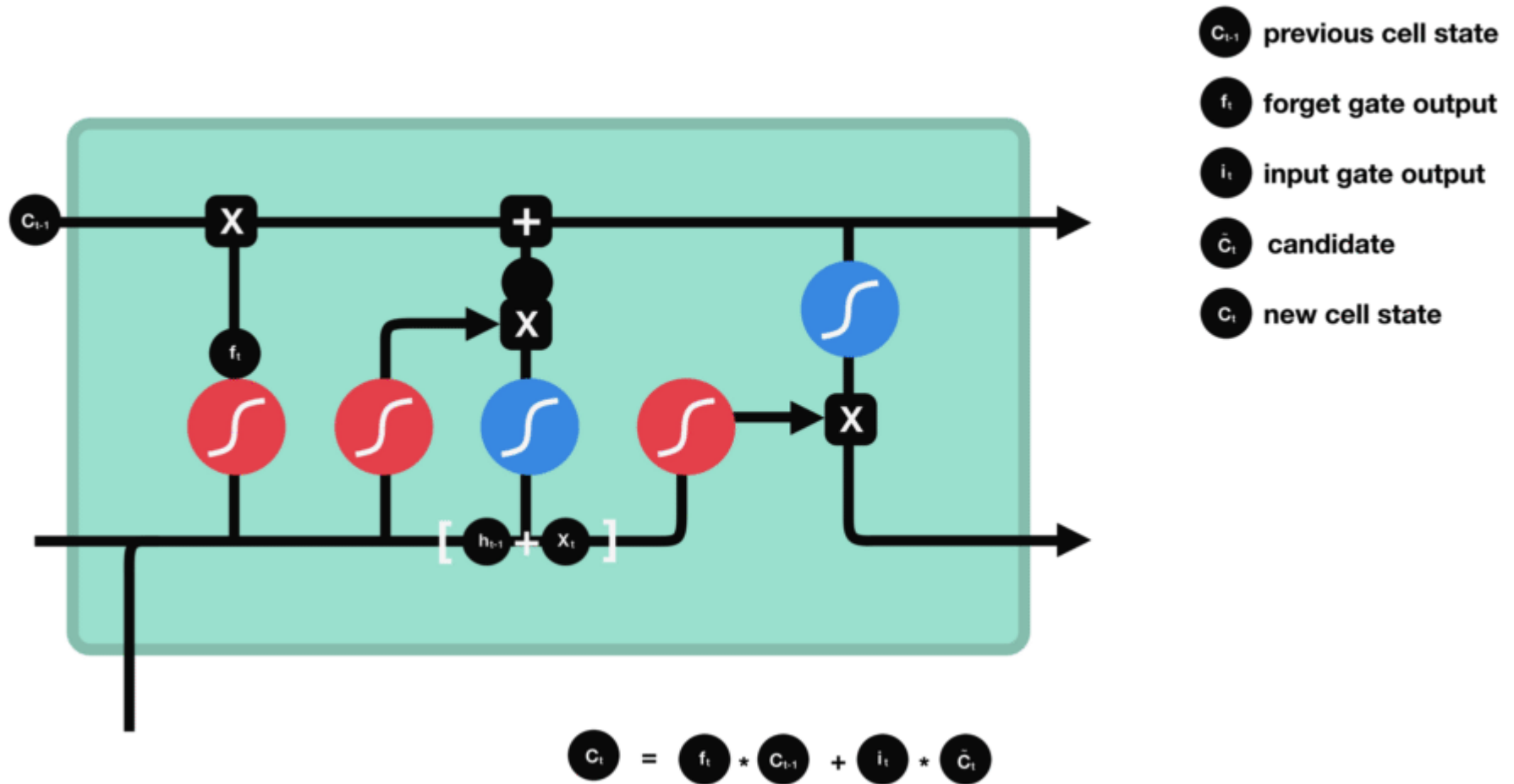
Long Short-Term Memory (LSTM)



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- **Cell State**
- Now we should have enough information to calculate the cell state.
- First, the **cell state** gets **pointwise multiplied** by the **forget vector**. This has a possibility of **dropping values in the cell state** if it gets multiplied by values near 0.
- Then we take the output from the **input gate** and do **a pointwise addition** which updates the cell state to new values that the neural network finds relevant.

Long Short-Term Memory (LSTM)



<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

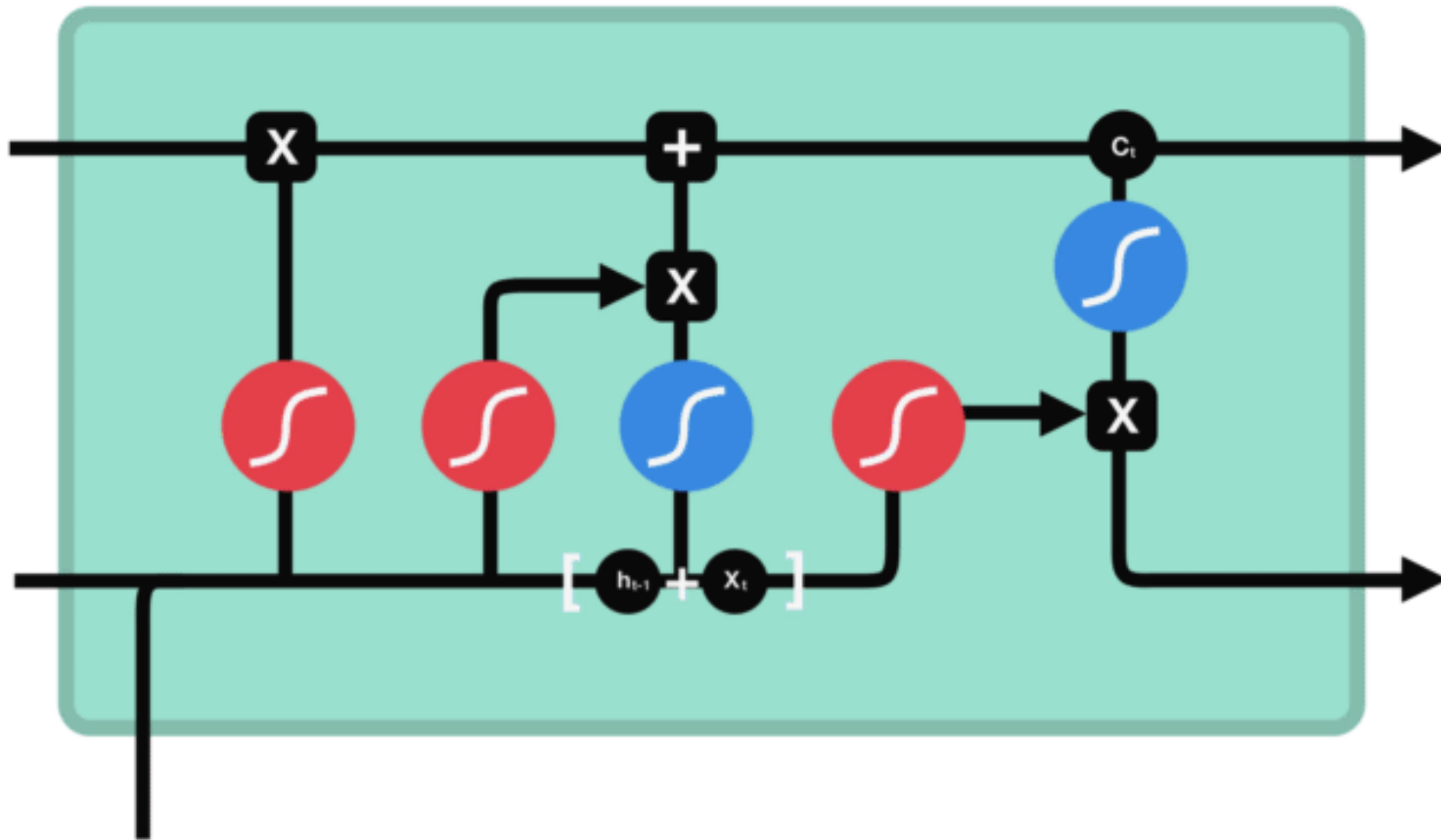
- Output Gate
- The output gate decides what the next hidden state should be. Remember that the hidden state contains information on **previous inputs**. The hidden state is also used **for predictions**.
- First, we pass the previous hidden state and the current input into a sigmoid function.
- Then we pass the newly modified cell state to the tanh function.
- We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- Output Gate
- The output gate decides what the next hidden state should be. Remember that the hidden state contains information on **previous inputs**. The hidden state is also used **for predictions**.
- First, we pass the previous hidden state and the current input into a sigmoid function.
- Then we pass the newly modified cell state to the tanh function.
- We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

Long Short-Term Memory (LSTM)



- c_{t-1} previous cell state
- f_t forget gate output
- i_t input gate output
- $\tilde{c}_t$ candidate
- c_t new cell state
- o_t output gate output
- h_t hidden state

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

- LSTM and GRU introduce additional mechanisms to RNNs that allow them to selectively **retain or update information over time**. This enables them to capture **long-term dependencies** in sequences and make use of relevant information from the past.

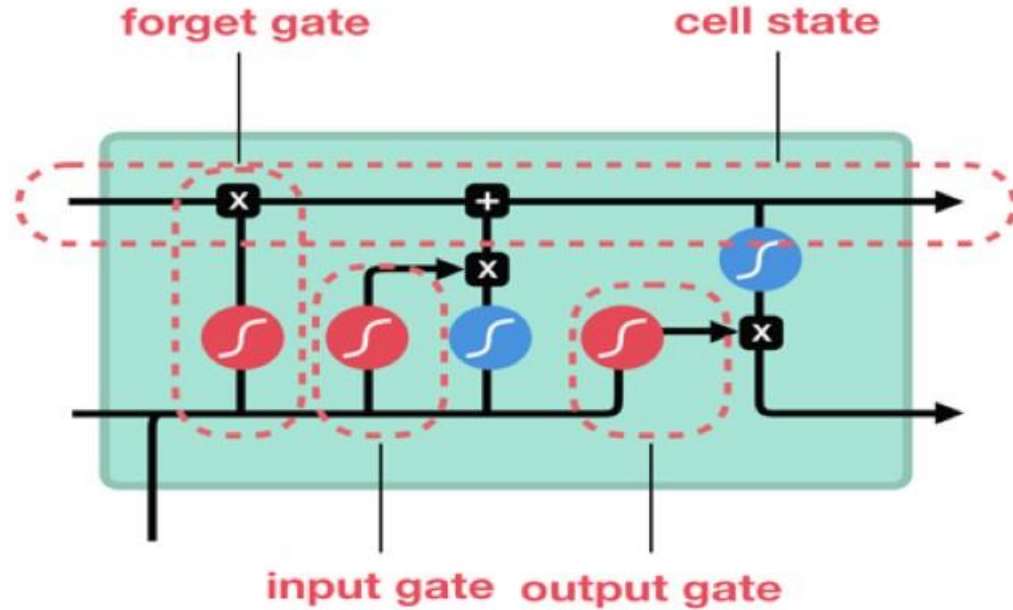
- LSTM was introduced by Hochreiter and Schmidhuber in 1997.
- It consists of memory cells that can store and **access information over long periods of time.**
- The key components of an LSTM cell are the **input gate, forget gate, output gate, and memory cell.**
- The input gate determines whether to update the memory cell with new information.
- The forget gate decides which information to discard from the memory cell.
- The output gate controls the information to be output from the memory cell.

GRU stands for Gated Recurrent Unit.(GRU)

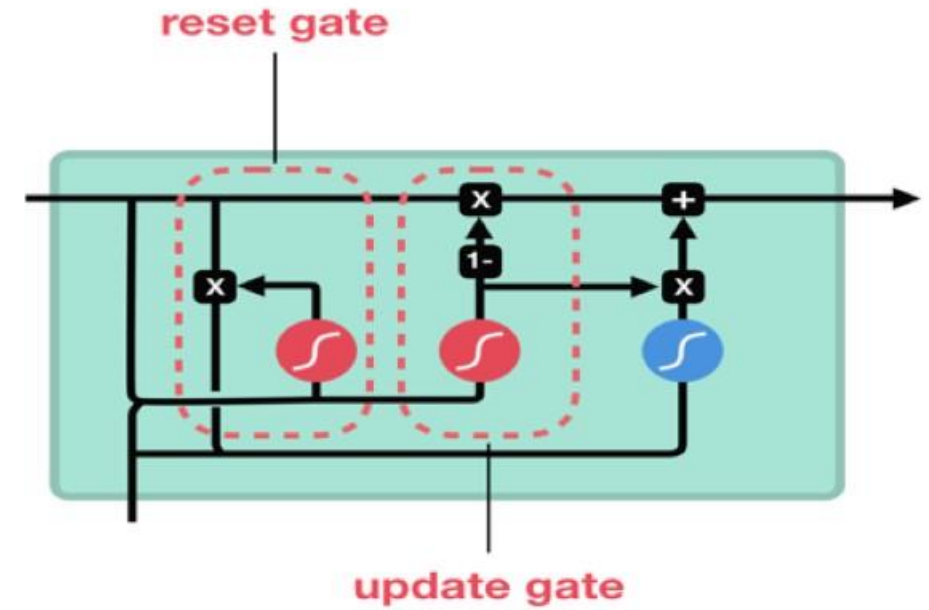
- GRU was introduced by Cho et al. in 2014 as a simplified version of LSTM.
- It combines the forget and input gates of LSTM into a single update gate.
- The reset gate determines the amount of information from the past to forget.
- GRU has a more compact architecture compared to LSTM, which makes it easier to train and computationally more efficient.

GRU vs LSTM

LSTM



GRU



sigmoid



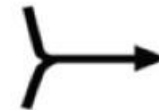
tanh



pointwise
multiplication



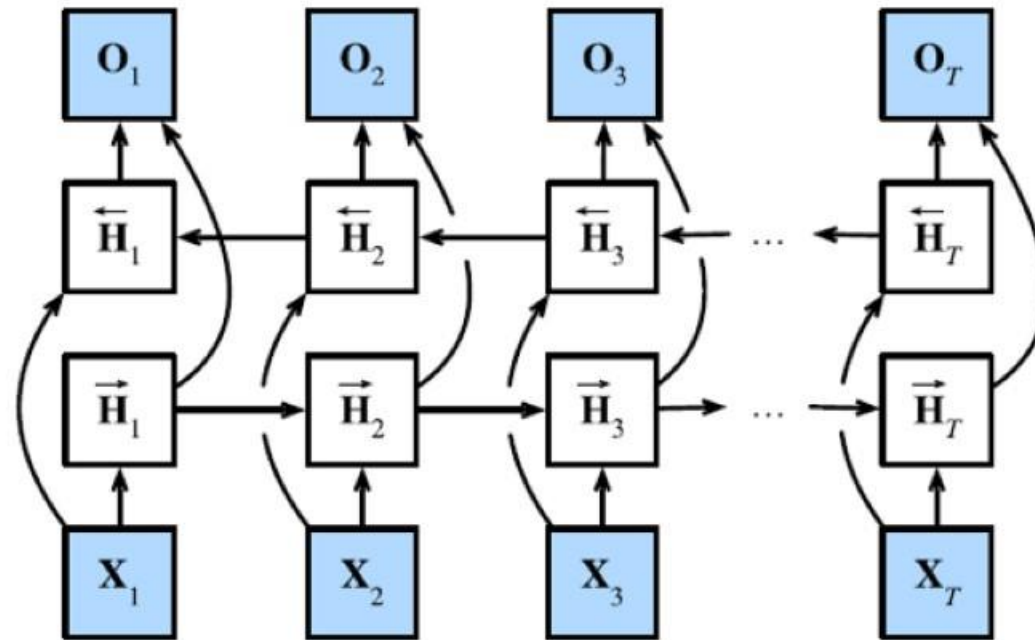
pointwise
addition



vector
concatenation

Bidirectional RNN (BiRNN)

- BiRNN processes the input sequence in both forward and backward directions using two separate hidden layers.



- Transformers are a more recent architecture that have gained significant attention, particularly with the introduction of the Transformer model in the "Attention is All You Need" paper by Vaswani et al. in 2017.
- Transformers are based on a **self-attention** mechanism that allows them to capture dependencies between elements in a sequence without the need for recurrent connections.
- Instead of maintaining a recurrent hidden state, transformers process the entire sequence in parallel, attending to different parts of the sequence using attention mechanisms. This **parallelization** makes transformers highly efficient for training and inference.

- Compared to RNNs, transformers have several advantages. They can better handle long-range dependencies, capture global context, and **parallelize computation**, making them more efficient for training on modern hardware.
- However, transformers may require **more data** to train effectively and can be **computationally expensive** due to the attention mechanism and the larger number of parameters compared to RNNs.

LAB 1

LAB 2



LECTURE SUMMARY

- ◉ RNN variants like LSTM and GRU have been developed to address the vanishing gradient problem and improve the modeling capabilities of RNNs. LSTM introduces gating mechanisms to control the flow of information, while GRU simplifies the gating mechanism.
- ◉ Transformers

Deep Learning Recent Updates

- Transformer-based architectures dominate most tasks
- Large Language Models (LLMs) with multi-task and in-context learning capabilities
- Diffusion models outperform GANs in image generation
- Efficient architectures: MobileNets, EfficientNet, and attention-based compression
- Self-supervised learning becoming the default training paradigm

- Multimodal models combining text, image, audio, and video
- Reinforcement learning from human feedback (RLHF) for alignment
- Neural scaling laws guide model size vs. performance trade-offs
- Low-rank adaptation (LoRA) and parameter-efficient fine-tuning methods
- Better optimization algorithms (Lion optimizer, AdamW improvements)

- Improved regularization and training stability techniques
- Edge AI and on-device deep learning acceleration
- Graph neural networks (GNNs) gaining traction in scientific and medical applications
- Time-series foundation models for healthcare and forecasting
- Explainable deep learning and improved interpretability tools



Thank You