Artificial Intelligence (Machine Learning & Deep Learning) [Course]

Week 8 – Deep Learning –

Recurrent Neural Networks (RNNs)

Long short-term memory (LSTM)

Gated Recurrent Unit Networks

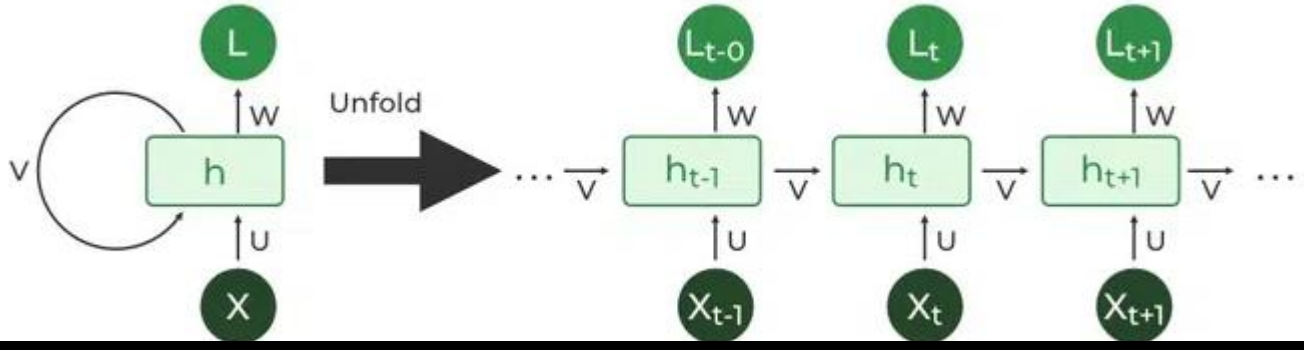
[See examples / code in GitHub code repository]

It is not about Theory, it is 20% Theory and 80% Practical – Technical/Development/Programming [Mostly Python based]

DL | What is a Recurrent neural networks (RNNs)?

Recurrent neural networks (RNNs) are a class of artificial neural networks designed for processing sequential data, such as text, speech, and time series,[1] where the order of elements is important. Unlike feed forward neural networks, which process inputs independently, RNNs utilize recurrent connections, where the output of a neuron at one time step is fed back as input to the network at the next time step. This enables RNNs to capture temporal dependencies and patterns within sequences.

RNNs allow the network to "remember" past information by feeding the output from one step into next step. This helps the network understand the context of what has already happened and make better predictions based on that. For example when predicting the next word in a sentence the RNN uses the previous words to help decide what word is most likely to come next.



This image showcases the basic architecture of RNN and the feedback loop mechanism where the output is passed back as input for the next time step.

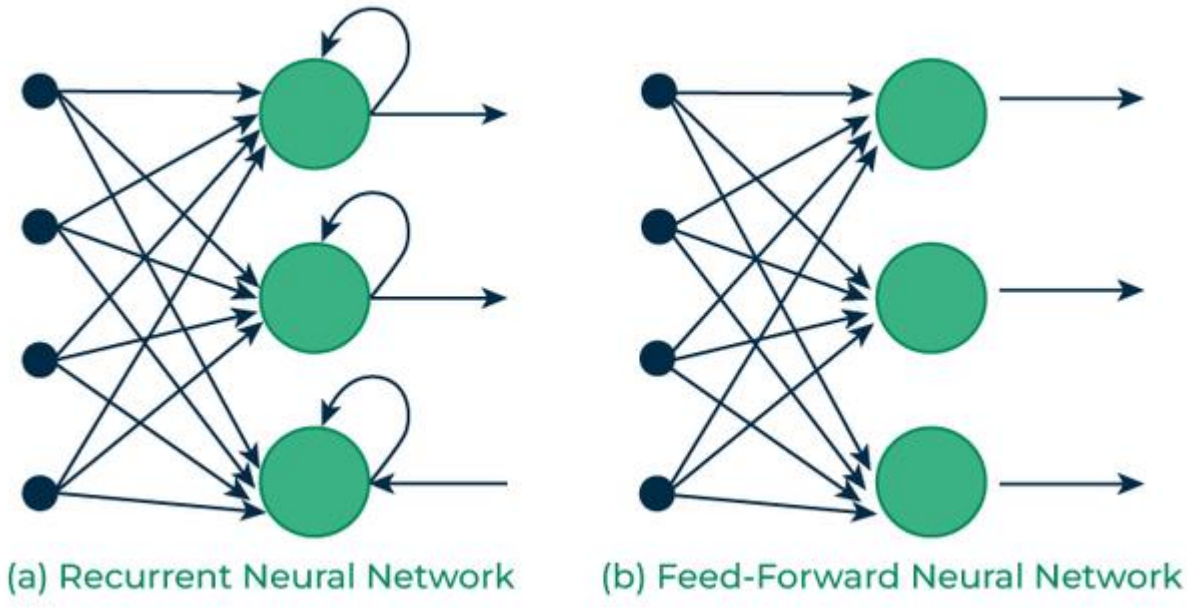
Reference:

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

https://www.ibm.com/think/topics/recurrent-neural-networks

DL | How RNN Differs from Feedforward Neural Networks?

Feedforward Neural Networks (FNNs) process data in one direction from input to output without retaining information from previous inputs. This makes them suitable for tasks with independent inputs like image classification. However FNNs struggle with sequential data since they lack memory. Recurrent Neural Networks (RNNs) solve this **by incorporating loops that allow information from previous steps to be fed back into the network**. This feedback enables RNNs to remember prior inputs making them ideal for tasks where context is important.



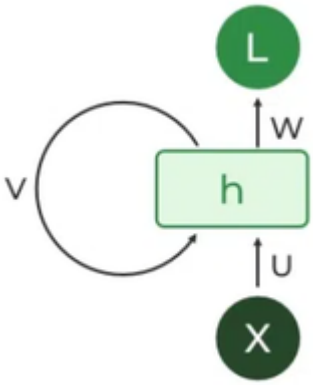
(a) Recurrent Neural Network

(b) Feed-Forward Neural Network

DL | Key Components of RNNs

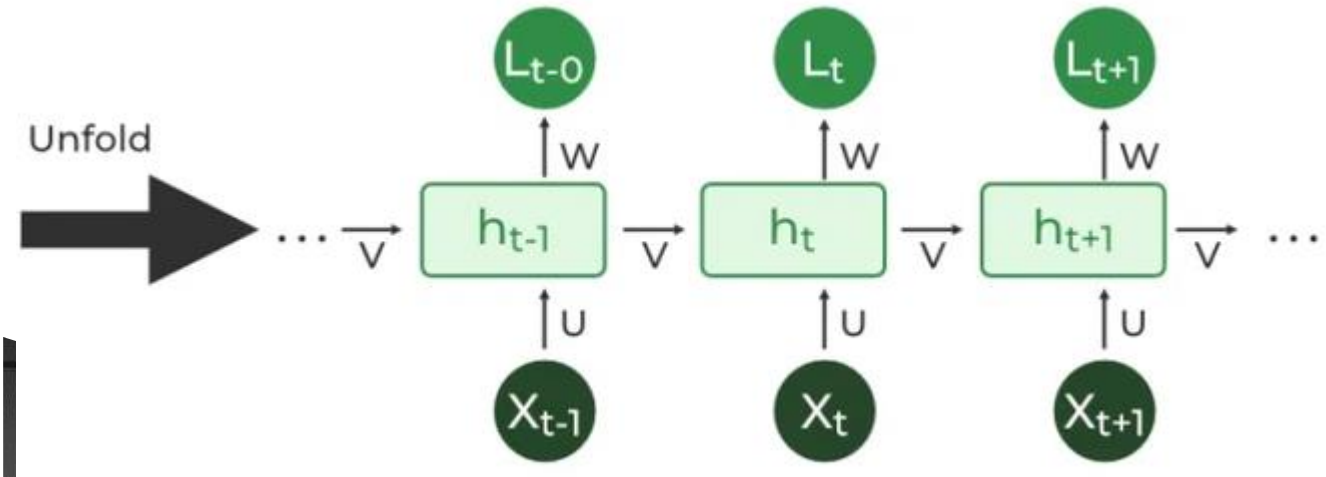
1. Recurrent Neurons

The fundamental processing unit in RNN is a **Recurrent Unit.** Recurrent units hold a hidden state that maintains information about previous inputs in a sequence. Recurrent units can "remember" information from prior steps by feeding back their hidden state, allowing them to capture dependencies across time.



2. RNN Unfolding

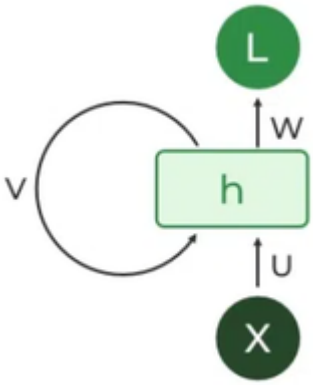
RNN unfolding or unrolling is the process of expanding the recurrent structure over time steps. During unfolding each step of the sequence is represented as a separate layer in a series illustrating how information flows across each time step.
This unrolling enables **backpropagation through time (BPTT)** a learning process where errors are propagated across time steps to adjust the network's weights enhancing the RNN's ability to learn dependencies within sequential data.



DL | Key Components of RNNs

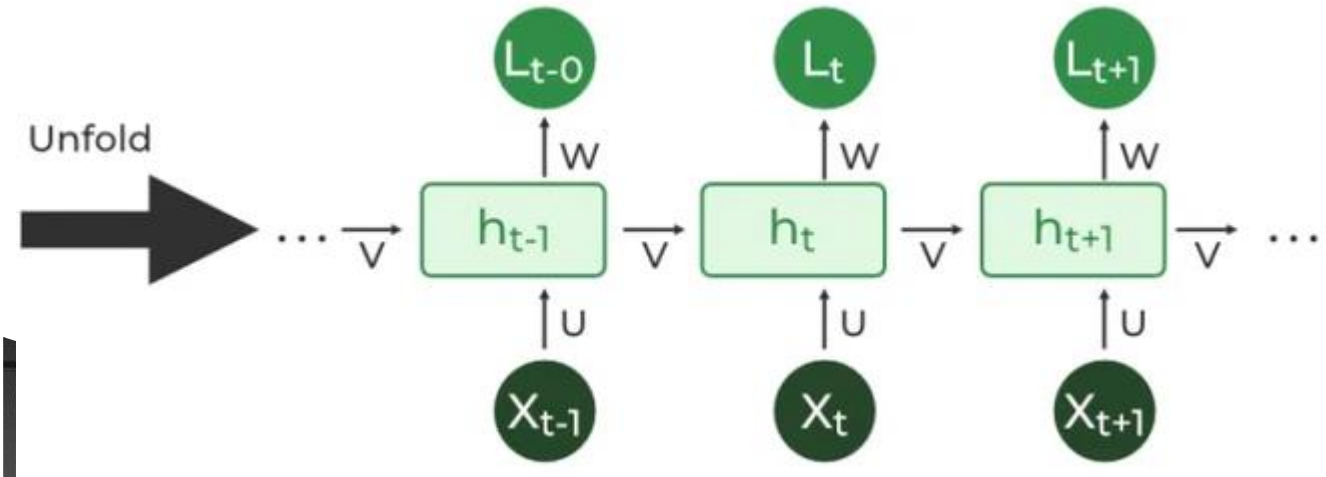
1. Recurrent Neurons

The fundamental processing unit in RNN is a **Recurrent Unit.** Recurrent units hold a hidden state that maintains information about previous inputs in a sequence. Recurrent units can "remember" information from prior steps by feeding back their hidden state, allowing them to capture dependencies across time.



2. RNN Unfolding

RNN unfolding or unrolling is the process of expanding the recurrent structure over time steps. During unfolding each step of the sequence is represented as a separate layer in a series illustrating how information flows across each time step.
This unrolling enables **backpropagation through time (BPTT)** a learning process where errors are propagated across time steps to adjust the network's weights enhancing the RNN's ability to learn dependencies within sequential data.



DL | Limitations of Recurrent Neural Networks (RNNs)

While RNNs excel at handling sequential data, they face two main training challenges i.e., vanishing gradient and exploding gradient problem:

Vanishing Gradient: During backpropagation, gradients diminish as they pass through each time step, leading to minimal weight updates. This limits the RNN's ability to learn long-term dependencies, which is crucial for tasks like language translation.

Exploding Gradient: Sometimes, gradients grow uncontrollably, causing excessively large weight updates that destabilize training. Gradient clipping is a common technique to manage this issue.
These challenges can hinder the performance of standard RNNs on complex, long-sequence tasks.

What Is a Gradient?
A gradient is a derivative of a function that has more than one input variable.
Reference: https://machinelearningmastery.com/gradient-in-machine-learning/

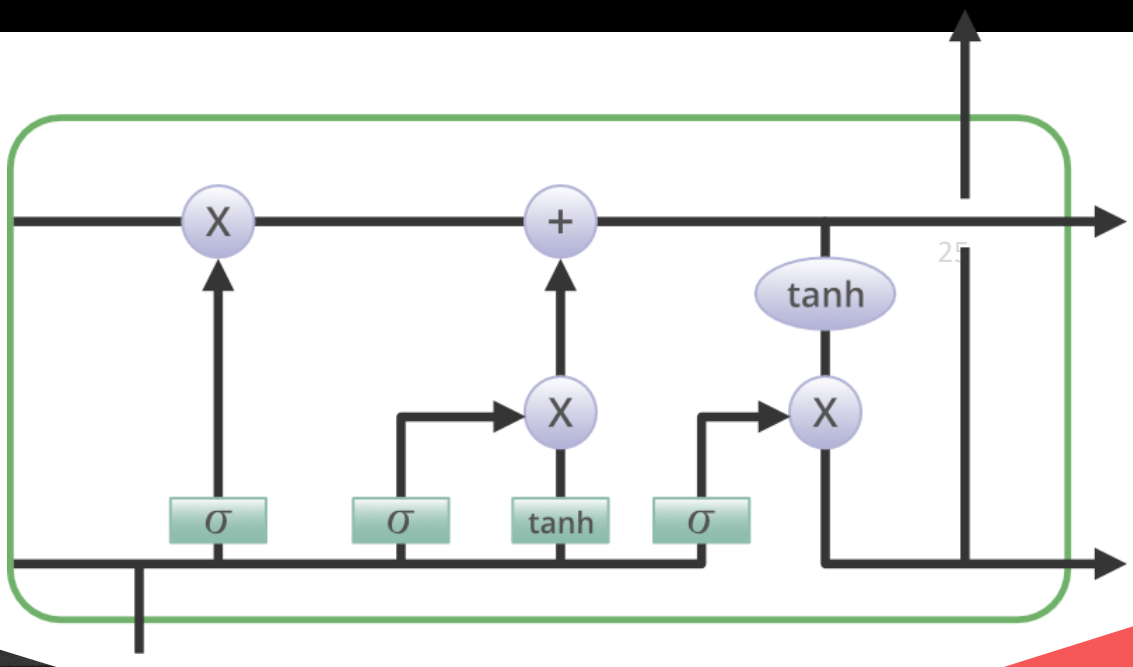
DL | Long Short-Term Memory - LSTM

LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. It fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such “long-term dependencies”.
There is no finer control over which part of the context needs to be carried forward and how much of the past needs to be ‘forgotten’.
Other issues with RNNs are exploding and vanishing gradients (explained later) which occur during the training process of a network through backtracking.

Thus, Long Short-Term Memory (LSTM) was brought into the picture. It has been so designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered. Long-time lags in certain problems are bridged using LSTMs which also handle noise, distributed representations, and continuous values.

DL | Structure of LSTM

The basic difference between the architectures of RNNs and LSTMs is that the hidden layer of LSTM is a gated unit or gated cell. It consists of four layers that interact with one another in a way to produce the output of that cell along with the cell state. These two things are then passed onto the next hidden layer. Unlike RNNs which have got only a single neural net layer of tanh, LSTMs comprise three logistic sigmoid gates and one tanh layer. Gates have been introduced in order to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part is to be discarded. The output is usually in the range of 0-1 where '0' means 'reject all' and '1' means 'include all'.



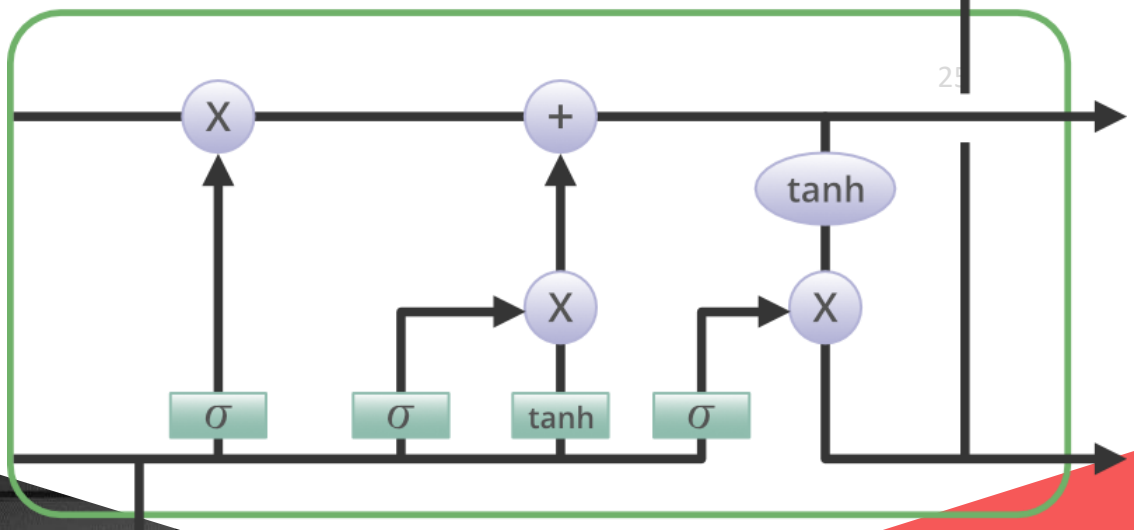
DL | Structure of LSTM

An LSTM unit consists of a memory cell and three gates that regulate the flow of information:

Input Gate: Controls what information is added to the memory cell.

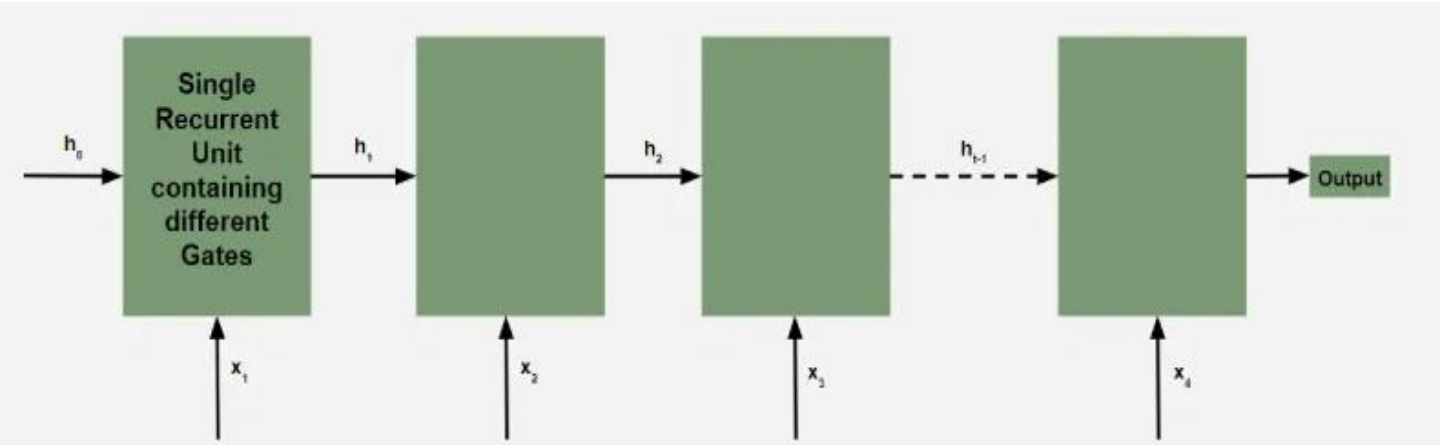
Forget Gate: Determines what information is discarded from the memory cell.

Output Gate: Controls what information is output from the memory cell.
These gates allow LSTMs to selectively retain or discard information, enabling them to maintain relevant long-term dependencies while processing sequential data.



DL | Gated Recurrent Units (GRU)

Gated Recurrent Units (GRUs) are a type of RNN introduced by Cho et al. in 2014. The core idea behind GRUs is to use **gating mechanisms** to selectively update the hidden state at each time step allowing them to remember important information while discarding irrelevant details. GRUs aim to simplify the LSTM architecture by merging some of its components and focusing on just two main gates: the **update gate** and the **reset gate**.



The GRU consists of **two main gates**:

1.Update Gate (*ztzt*): This gate decides how much information from previous hidden state should be retained for the next time step.

2.Reset Gate (*rtrt*): This gate determines how much of the past hidden state should be forgotten.

These gates allow GRU to control the flow of information in a more efficient manner compared to traditional RNNs which solely rely on hidden state.

DL | GRU vs LSTM

GRUs are more computationally efficient because they combine the forget and input gates into a single update gate. GRUs do not maintain an internal cell state as LSTMs do, instead they store information directly in the hidden state making them simpler and faster.

Feature	LSTM (Long Short-Term Memory)	GRU (Gated Recurrent Unit)
Gates	3 (Input, Forget, Output)	2 (Update, Reset)
Cell State	Yes it has cell state	No (Hidden state only)
Training Speed	Slower due to complexity	Faster due to simpler architecture
Computational Load	Higher due to more gates and parameters	Lower due to fewer gates and parameters
Performance	Often better in tasks requiring long-term memory	Performs similarly in many tasks with less complexity

Reference:
https://www.geeksforgeeks.org/gated-recurrent-unit-networks/

DL | How GRUs Solve the Vanishing Gradient Problem

Like LSTMs, GRUs were designed to address the **vanishing gradient problem** which is common in traditional RNNs. GRUs help mitigate this issue by using gates that regulate the flow of gradients during training ensuring that important information is preserved and that gradients do not shrink excessively over time. By using these gates, GRUs maintain a balance between remembering important past information and learning new, relevant data.

Reference:

https://www.geeksforgeeks.org/gated-recurrent-unit-networks/

DL| Recurrent Neural Networks (RNNs)
Long short-term memory (LSTM)
Gated Recurrent Unit Networks - Exercise

See code here: https://github.com/ShahzadSarwar10/AI-ML-Explorer/blob/main/Week8/Case8-3-TF-Recurrent-Neural-Networks-RNN.py

https://github.com/ShahzadSarwar10/AI-ML-Explorer/blob/main/Week8/Case8-4-TF-LongShortTerm-Memory-LSTM-RNN.py

https://github.com/ShahzadSarwar10/AI-ML-Explorer/blob/main/Week8/Case8-5-TF-GatedRecurrentUnitNetworks.py

You should be able to analyze – each code statement, you should be able to see trace information – at each step of debugging. "DEBUGGING IS BEST STRATEGY TO LEARN A LANAGUAGE." So debug code files, line by line, analyze the values of variable – changing at each code statement. BEST STRATEGY TO LEARN DEEP.

Let's put best efforts.
Thanks.
Shahzad – Your AI – ML Instructor

Thank you - for listening and participating

- ❑**Questions / Queries**
- ❑**Suggestions/Recommendation**
- ❑**Ideas…..?**

Shahzad Sarwar
Cognitive Convergence
https://cognitiveconvergence.com
shahzad@cognitiveconvergence.com
voice: +1 4242530744 (USA) +92-3004762901 (Pak)