NLP FOR DL

→ NLP FOR ML → OHE, BOW, TFIDF; WORD2VEC; AVG WORD2VEC

→ NLP FOR DL → NLP In Deep Learning [Generative AI → LLM, MultiModal]

① Simple RNN → LSTM / GRU RNN → Bidirectional RNN → Encoder Decoder

Encoder Decoder → Self Attention → Transformers → Gen AI

RNN — maintains Order / maintains Past Memory

is a neural network to handle sequential Data by using memory of past inputs through a hidden state

Imp Sequential Data = Data in sequence (Order is imp.)

e.g.:
- Timeseries Data (1 Feb – 30 Feb)
- Text (I love Deep Learning)
- Audio

APPLICATION

- Chat bot App → Q & A
- Language Translation → Eng → French
- Text Generation → A sentence → Completion of sentences
- Auto Suggestion → LinkedIn, Gmail
- Time Series → Sales Data Future Prediction.

Q.) Can we solve Sequential with ANN?

Text	O/P
The food is good	1
The food is bad	0
The food is not good	0

vocabulary = 4

BOW

	food	good	bad	not
S1	[1,	1	0	0]
S2	[1	0	1	0]
S3	[1	1	0	1]

Summary:

- ✅ ANN *can* handle sequential data (technically)
- ❌ But **not ideal** for learning patterns over time
- ✅ Use **RNNs, LSTMs, GRUs, or Transformers** for best results with sequential data

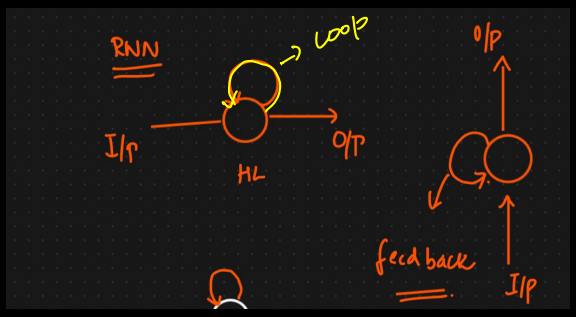
{ Because context meaning is lost

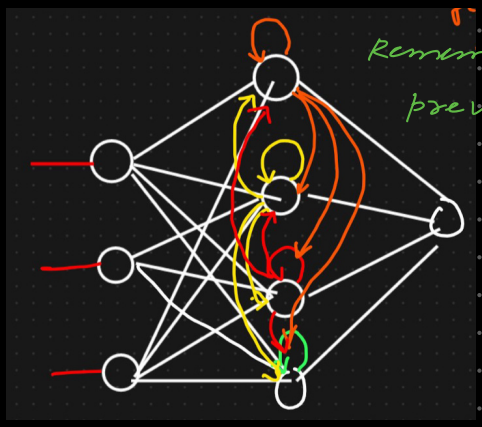
vv imp. : RNN's are sequentially aware.
: Maintain hidden state (for past memory)

🔶 Difference in How They Handle Sequence

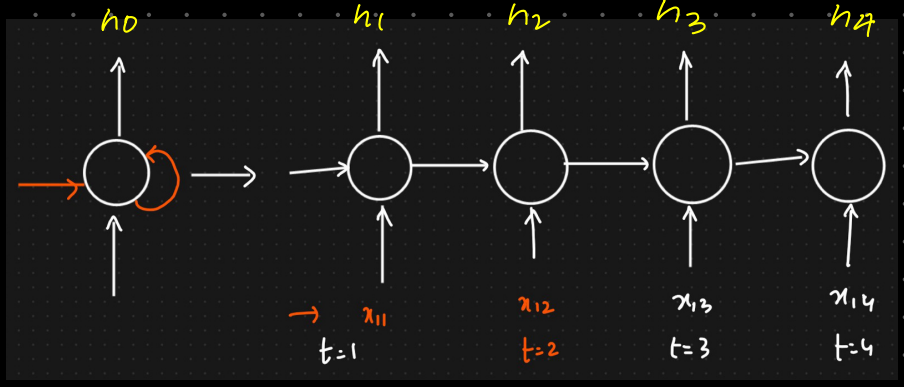
Feature	ANN	RNN
Sequence Awareness	❌ No built-in awareness	✅ Processes input in **order**
Memory of Past Inputs	❌ None	✅ Maintains **hidden state**
Time Step Handling	❌ Must be encoded manually	✅ Each step processed **sequentially**
Best For	Static data (images, tabular)	Time series, language, speech

Simple RNN — Self feedback loop.





Steps:



x_{11} x_{12} x_{13} x_{14}
The Food is Good

Imp : Previous outputs are shared with other neurons so previous content is maintained.

RNN Steps

RNN steps are similar to ANN but there is one more ingredient, (past memory) or previous timestamp memory

h_t = current
h_{t-1} = past
Previous inputs

Step-by-step (Simple RNN, timestep t)

1 Input layer

- Take current input x_t (e.g., word embedding vector).

2 Weighted sum from current input

- Multiply by input-to-hidden weights:

$$W_x x_t$$

3 Weighted sum from previous hidden state (memory)

- Multiply the previous hidden state h_{t-1} by hidden-to-hidden weights:

$$W_h h_{t-1}$$

4 Combine + bias

- Add them together with bias b_h:

$$a_t = W_x x_t + W_h h_{t-1} + b_h$$

5 Activation for hidden state (e.g., tanh or ReLU)

- Get the new hidden state:

$$h_t = \tanh(a_t)$$

6 Output layer

- Multiply by hidden-to-output weights:

$$z_t = W_y h_t + b_y$$

7 Output activation (depends on task)

- Classification → softmax
- Binary → sigmoid
- Regression → none (linear)

Important

Big picture difference from ANN:

- ANN: Output depends only on current input.
- RNN: Output depends on current input + previous hidden state (memory).

V. V. Imp →

Simple RNN can only remember information for few current time stamps before it fades away.