RNN Notes
By Shivani Bhatia

⇒ RNN designed for sequential data. In, sequential data, entities occur in particular order. If you break the order, you don't have meaningful sequence anymore.

⇒ Ex of sequential data : Videos, Peice of music etc.

⇒ RNN variant of vanilla neural network tailared to learn sequential data.

⇒ Main difference b/w normal neural nets & RNN is that RNN's have two dimensions - time t along with depth l (the usual layers).

⇒ In RNN, the output is said as, "the output at layer l and time t", notated as a^l_t, unlike normal neural net output which is given as a_t.

⇒ RNN is network that changes it's state with time. State of a^l_t changes to a^l_{t+1} as it's sequent element in the sequence.

⇒ The output of layer l at time t+1, a^l_{t+1} depends on 1) Output of previous layer at same time step a^{l-1}_{t+1} & 2) it's own previous state a^l_t:

$$a^l_{t+1} = g(a^{l-1}_{t+1}, a^l_t)$$

→ Feedforward equations are simply an extensions of vanilla neural nets : the only difference being that now there is weight associated with both a_{t+1}^{l-1} and a_t^l :

$$a_{t+1}^{l} = \sigma(W_F . a_{t+1}^{l-1} + W_R . a_t^l + b^l)$$

W_F → Feedforward weights

W_R → Recurrent weights

→ RNN allows network to remember past information by feeding output from one step into next step.

→ RNN's help network understand context of what already has happened & make better predictions based on that.

→ Feedback loop mechanism : Output passed back as input for next time step.

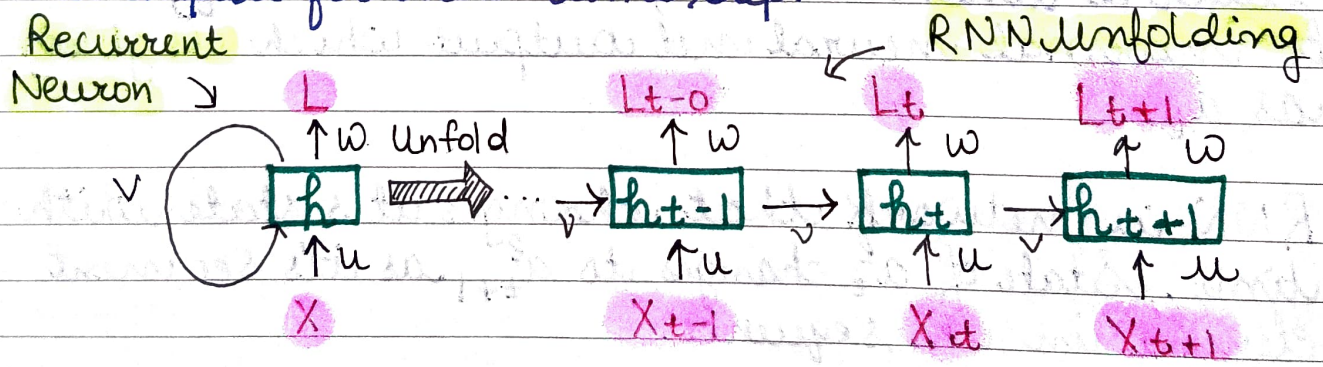


Fig : Basic architecture of RNN and feedback loop mech.

⇒ Feedforward Neural Network process data in one direction from input to output without retaining information from previous outputs. However, this is not the case with RNN.

⇒ FNN is suitable for tasks with independent inputs like image classification.

⇒ Key components of RNNs:

1) Recurrent Neurons:

⇒ Fundamental processing units in RNN is a Recurrent Unit (R.U.).

⇒ Recurrent unit holds a hidden state that maintains information about previous inputs in a sequence.

⇒ R.U can remember info. from prior steps by feeding back their hidden state, allowing them to capture dependancy over time.

2) RNN Unfolding:

⇒ Process of expanding recurrent structure over time steps.

⇒ During unfolding each step of the sequence is represented as separate layer in series illus-

trating how information flows across each time steps.

→ Unrolling enables Backpropogation through time (BPTT), a learning process, where errors are propogated across time steps to adjust the networks weights enhancing RNN's ability to learn dependencies with sequential data.

→ RNN Neural Network Architecture?

Unlike traditional deep neural networks, where each dense layer has distinct weight matrices, RNN use shared weight matrices across time stamps, allowing them to remember information over sequences.

→ In RNN's, we do following calculations:-

① Hidden State Calculation

$$h = \sigma(U \cdot X + W \cdot h_{t-1} + B)$$

- h: Current hidden state
- U, W: Weight matrices
- B: Bias

② Output Calculation

$$Y = O(V \cdot h + C)$$

- Y: Output
- O: activation function
- V: Weight
- C: Bias

(3.) Overall function

$$Y = f(X, R, W, U, V, B, C)$$

This funcxn. defines the entire RNN operation, where state matrix S holds each element s_i representing the network's state at each time step i.

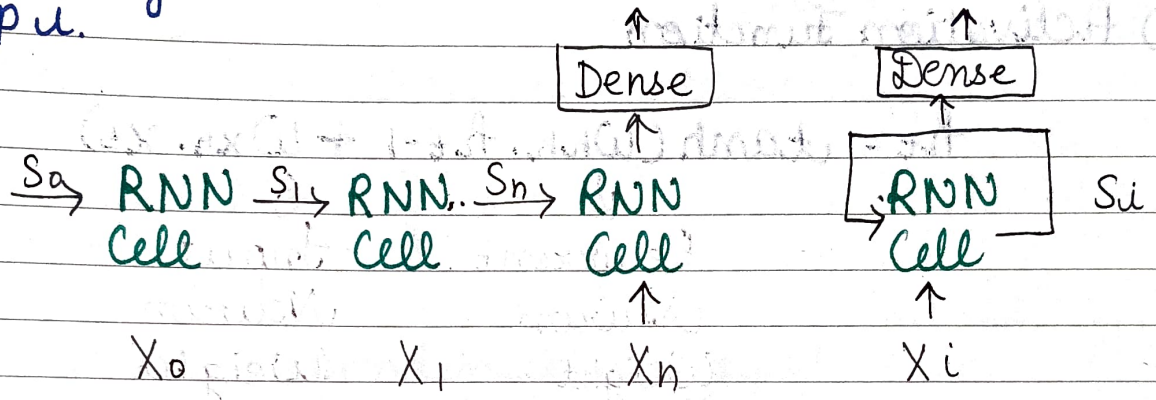


Fig: Recurrent Neural Architecture

→ RNN Working

At each time step RNNs process units with a fixed activation function. These units have an internal hidden state that acts as memory that retains information from previous time-steps. This memory allows the network to store past knowledge & adapt based on new inputs.

Updating hidden state in RNN's

1) State Update

$$h_t = f(h_{t-1}, x_t)$$

- h_t → Current State
- h_{t-1} → Previous state
- x_t → Input @ current time stamp

2) Activation Function

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

- W_{hh} → Recurrent Neuron Weight Matrix
- W_{xh} → Input Neuron Weight Matrix

3) Output Calculation

$$y_t = W_{hy} \cdot h_t$$

- W_{hy} → Weight @ output layer

★ These parameters are updated using backpropogation. Since, RNN works on sequential data, we call it BPTT.

→ BackPropagation through time in RNNs

→ Since RNN's process sequential data BPTT is used to update network's parameter.

→ The loss function L(θ) depends on final hidden state h_3 & each hidden state relies on preceding ones.

→ In BPTT gradients are backpropogated through each time step. Essential for updating network parameters based on temporal dependancy.

1) Simplified Gradient Calculation:

$$\frac{\partial L(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial h_3} \cdot \frac{\partial h_3}{\partial w}$$

2) Handling Dependencies in Layers:

$$h_3 = \sigma(w \cdot h_2 + b)$$

3) Gradient Calculation with Explicit & Implicit parts:

The gradient is broken down into explicit & implicit parts summing up indirect paths from each hidden state to weights.

$$\frac{\partial h_3}{\partial w} = \frac{\partial h_3^+}{\partial w} + \frac{\partial h_3}{\partial h_2} + \frac{\partial h_2^+}{\partial w}$$

4) Final Gradient Expression:

Final derivative of loss function w.r.t to weight matrix W is computed:

$$\frac{\partial L(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial h_3} \cdot \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial w}$$

→ Types of RNN

1.) One-to-One RNN
2.) One-to-many RNN
3.) Many-to-one RNN
4.) Many-to-Many RNN

1.) One-to-One RNN ÷ Single input & Single Output. Used for straightforward classification tasks such as binary classification with no sequential data involved.

2.) One-to-many RNN ÷ Single input to produce multiple outputs over time. One input triggers a sequence of predictions (output).

3.) Many-to-One RNN ÷ Recieves a sequence of inputs and generates a ~~sequence~~ single of outputs. Ex ÷ Sentiment model, recieve sequence of words but produce single output like positive, negative & neutral.

4.) Many-to-Many RNN ÷ Recieve sequence of input, generate sequence of outputs.

→ Variants of RNN ÷

(i) Vanilla RNN ÷ RNN consist of single hidden layers where weights are shared across time steps. Suitable for short term dependancy.

Bi-directional RNN ÷ Process input in both forward & backward directions, capturing both past and future time context for each time step.

LSTM's ÷ Intraduce memory mechanism to over come vanishing gradient. It has 3 gates,

1.) Input gate ÷ Control how much new info to add.
2.) Forget gate ÷ Decide what past info to discard.
3.) Output gate ÷ Regulates what info. should be output at the current step. Selective memory enables LSTM to handle long term dependency.

Gated Recurrent Unit ÷ less complex, combines input gate & forget gates into single update gate & streamlining output mechanism.

Vanishing vs Exploding Gradient

① Vanishing Gradient ÷ When training models over time, gradients can shrink as they pass through many steps. This makes it hard for model to learn long-term patterns since earlier information becomes almost irrelevant.

② Exploding Gradient ÷ Gradients grow too large, leading to instability. Makes difficult for model to learn properly.

⇒ Gated Recurrent Unit (GRU)

Two main gates :-

(i) Update Gate (z_t): Gate decides how much information from previous hidden state should be retained for next time step.

(ii) Reset Gate (r_t) : How much past hidden state should be forgotten is decided here.

→ Equations for GRU operations

Internal workings of a GRU

① Reset Gate

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

② Update Gate

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

③ Candidate Hidden State

$$h_t' = \tanh(W_h \cdot [r_t \cdot h_{t-1}, x_t])$$

h_t' → Potential New hidden state

h_{t-1} → Old hidden state

④ Hidden State

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h'_t$$

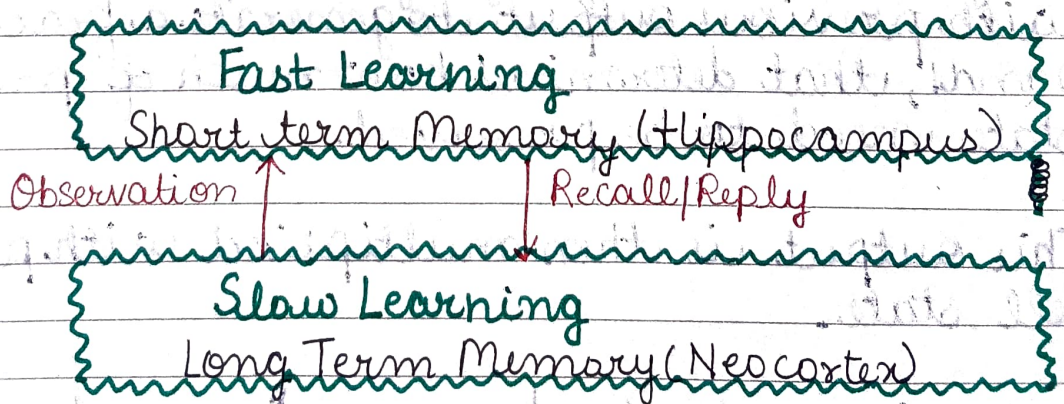
↳ Final hidden state

Final hidden state is weighted average of previous hidden state h_{t-1} & candidate hidden state h'_t & updated gate z_t.

→ Short-term memory

Short term memory is a cognitive system that allows individuals to hold and process information for a short period of time.

Capacity of short-term memory is limited, usually around 7(+-2) items and duration of short-term memory is around 20-30 seconds.



Long Short Term Memory Network (LSTM)

→ LSTM designed by Hochreiter & Schimidhuber.

→ LSTM recurrent unit tries to "remember" all the past knowledge that network is seen so far and to "forgot" irrelevant data.

→ This is done by introducing different activation function layers called "gates" for different purposes

→ Internal Cell State :- Each LSTM recurrent unit maintain a vector called Internal Cell State which conceptually describes the information that was chosen to be retained by the previous LSTM recurrent unit

→ 4 Different Gates in LSTM :-

1) Forget Gate (F) :- At forget gate input is combined with previous output to generate a fraction b/w 0 and 1, that determines how much of previous state need to be preserved.

This output is then multiplied with previous cell state.

Activation output = 1, means "remember everything"
Activation output = 0, means "forget everything"

2.) Input Gate (i) :- Operates on same signal as forget gate, but objective is to decide which new information is going to enter state of LSTM. The output of input gate is multiplied with output of tanh block that produces the new value that must be added to previous state.

3.) Input Modulation Gate (g) :- Sub-part of input gate.
Used to modulate information that the input gate will write onto the internal state cell by adding non-linearity to the information and making the information Zero-mean.
Done to reduce learning time as Zero-mean input has faster convergence.

4.) Output gate (O) :- At output gate, the input & previous state are gated as before to generate another scaling fraction that is combined with the output of tanh block that brings the current state. The output is then given out. The output and state are fed back into LSTM block.

→ Working of LSTM

① Take input the current input, previous hidden state, and previous internal cell state.

② Calculating value of 4 different gates :-

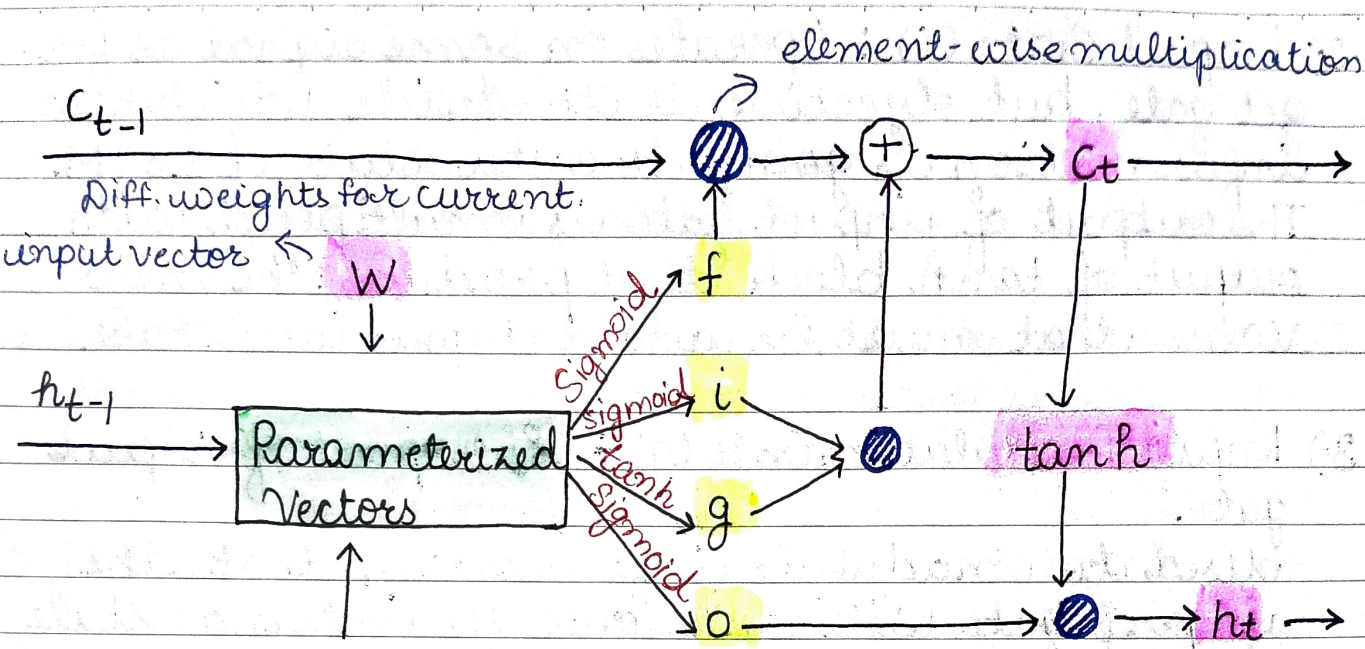


Fig: LSTM Working

- For each gate, calculate parameterized vectors for current Input and previous hidden state by element-wise multiplication with concerned vector with respective weights for each gate.

- Apply respective activation function for each gate element-wise on parameterized vectors. Below given is list of gates & activation funcxn. applied to it.

③ Calculate current internal cell state by first calculating element-wise multiplication vector of input gate and input modulation gate, then calculate element-wise multiplication vector of the forget gate & previous internal cell state & then add the two vectors.

$$C_t = i \odot g + f \odot C_{t-1}$$

④ Calculate the current hidden state by first taking the element-wise hyperbolic tangent of current internal cell state vector and then performing element-wise multiplication with output gate.