RNN

Recurrent neural networks (RNNs) are a type of neural network architecture mainly used to detect patterns in a data sequence.

Common Interview Questions:

1. What is a recurrent neural network (RNN)?
2. What are the advantages and disadvantages of using RNNs?
3. How do RNNs handle long-term dependencies?
4. What types of problems are best suited for RNNs?
5. What are the different types of RNNs?
6. How can RNNs be used for sequence prediction?
7. How can RNNs be used for natural language processing?
8. What are the challenges of training RNNs?
9. How can RNNs be used for action recognition?
10. List any real-time applications of RNN.

RNNs: Key Concepts

- RNN
- LSTM
- GRU
- Vanishing Gradient
- Clipping Gradient
- BPTT

Key RNN Concepts

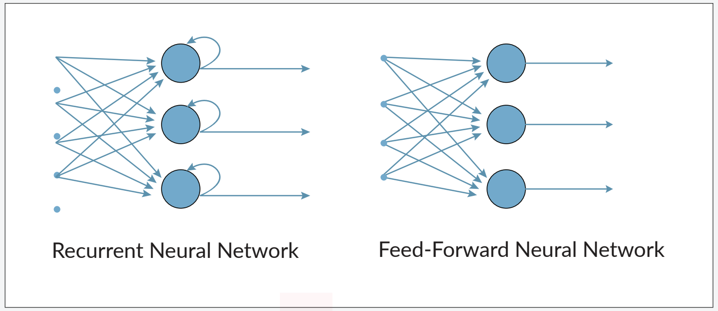
A recurrent neural network has a memory of past experiences. The recurrent connection preserves these experiences and helps the network keep a notion of context. Output vectors' contents are influenced not only by the input you just feed into them but also by the entire history of inputs you have fed them in the past.

RNNs

- Perform well when the input data is interdependent in a sequential pattern
- Correlation between the previous input and the next input
- Introduce bias based on your previous output

CNN/FF-Nets

- All the outputs are self-dependent.
- Feed-forward nets do not remember historic input data at test time unlike recurrent networks.



How RNN Works ?

- TTake the sequential input of any length.
- Apply the same weights to each step.
- Can optionally produce output on each step

LSTM, GRU Variations

LSTM, GRU - RNN Variations

- LSTMs and GRUs were created as the solution to short-term memory.
- LSTMs and GRUs can be found in speech recognition, speech synthesis, and text generation.
- It can learn to retain only relevant information to make predictions and forget non-relevant data.
- Like a conveyor belt, it runs through the entire chain with minor interactions.
- LSTM does have the ability to remove/add information to cell states regulated by structures called gates.

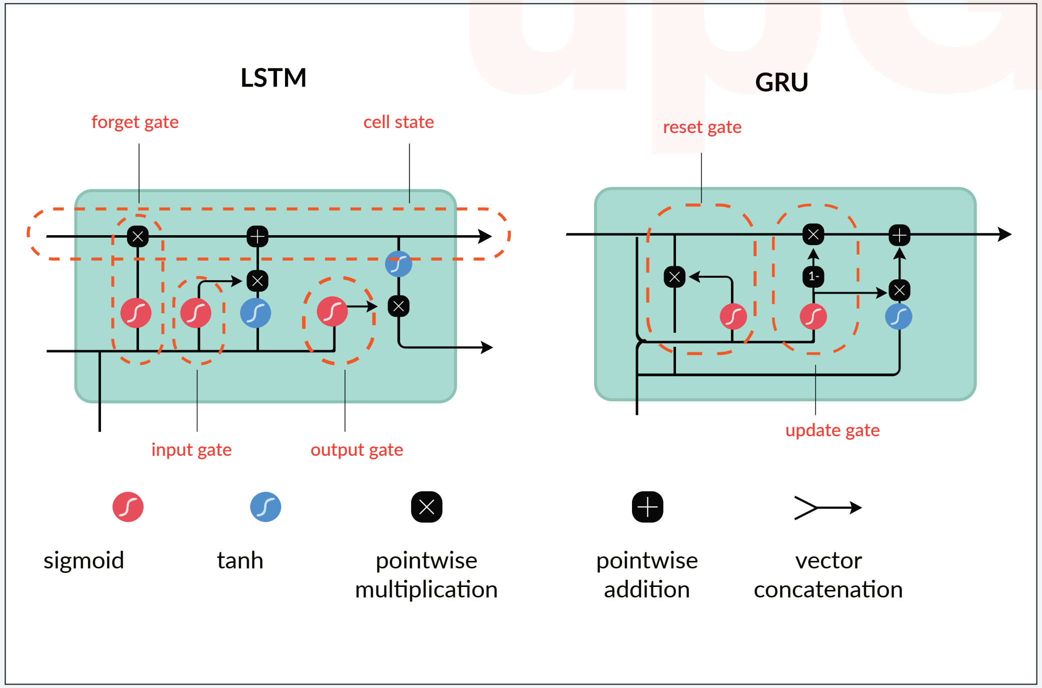
LSTM Return States After Computation

Default: Last Hidden State (Hidden State of the last time step)

return_sequences=True : All Hidden States (Hidden State of ALL the time steps)

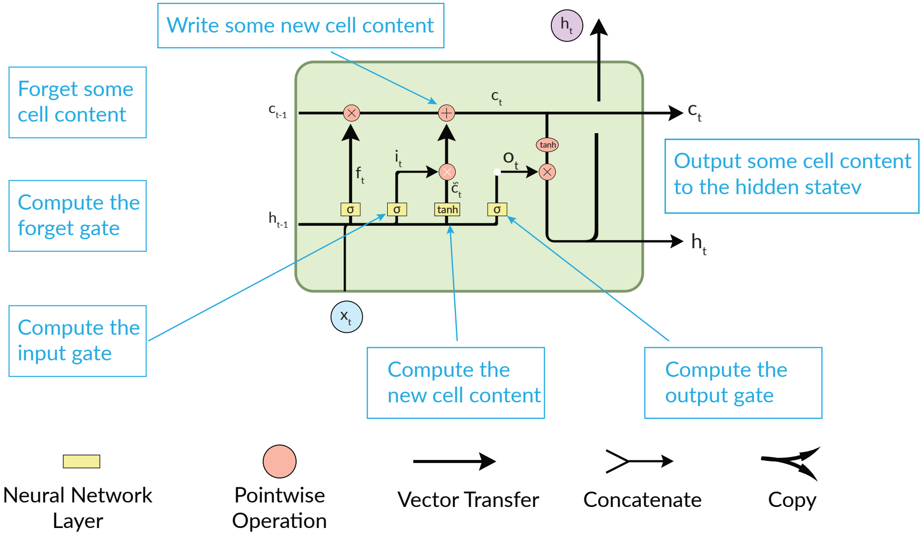
return_state=True : Last Hidden State+ Last Hidden State (again!) + Last Cell State (Cell State of the last time step)

return_sequences=True + return_state=True: All Hidden States (Hidden State of ALL the time steps) + Last Hidden State + Last Cell State (Cell State of the last time step)



Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



FFN vs RNN Network

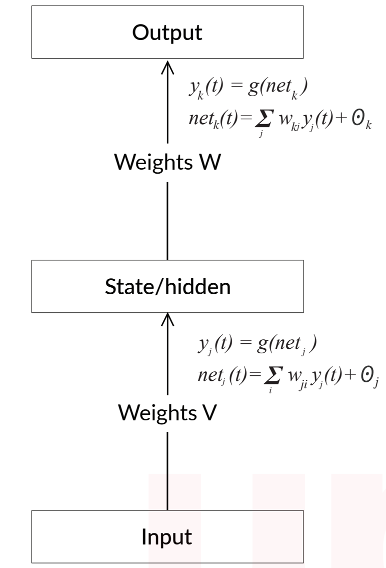

Output
$y_k(t) = g(net_k)$
$net_k(t)=\sum_j w_{kj} y_j(t)+\Theta_k$
Weights W
State/hidden
$y_j(t) = g(net_j)$
$net_j(t)=\sum_i w_{ji} y_j(t)+\Theta_j$
Weights V
Input

Figure 1: A feedforward network.

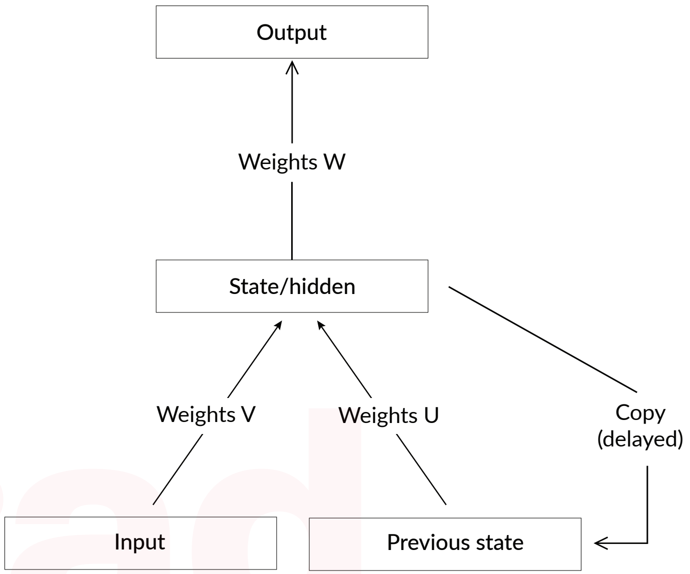
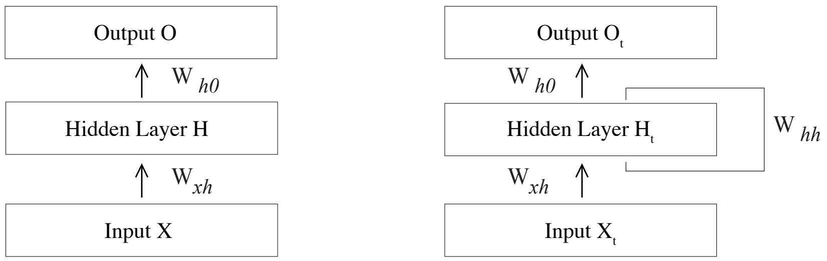


Figure 1: A feedforward network.



Feedforward Neural Network

Recurrent Neural Network

Figure 1: Visualisation of differences between Feedfoward NNs und Recurrent NNs

While Feedforward Networks pass information through the network without cycles, the RNN has cycles and transmits information back into itself.
This enables them to extend the functionality of Feedforward
Networks to also take into account previous inputs X0:t−1 and not only the current input Xt.

FFN vs RNN Network

Long Short-Term Memory (LSTM)

We have a sequence of inputs x(t), and we will compute a sequence of hidden states h(t) and cell states c(t). On timestep t:

Sigmoid function: all gate values are between 0 and 1

Forget gate: controls what is kept vs forgotten, from previous cell state

$$f^{(t)} = \sigma \left(W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

Input gate: controls what parts of the new cell content are written to cell

$$i^{(t)} = \sigma \left(W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

Output gate: controls what parts of cell are output to hidden state

$$o^{(t)} = \sigma \left(W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

New cell content: this is the new content to be written to the cell

$$\tilde{c}^{(t)} = tanh \left(W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

$$c = f^{(t)} \, o \, c^{(t-1)} + i^{(t)} \tilde{c}^{(t)}$$

Hidden state: read ("output") some content from the cell

$$h^{(t)} = o^{(t)} \, o \; tanh \; c^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise product

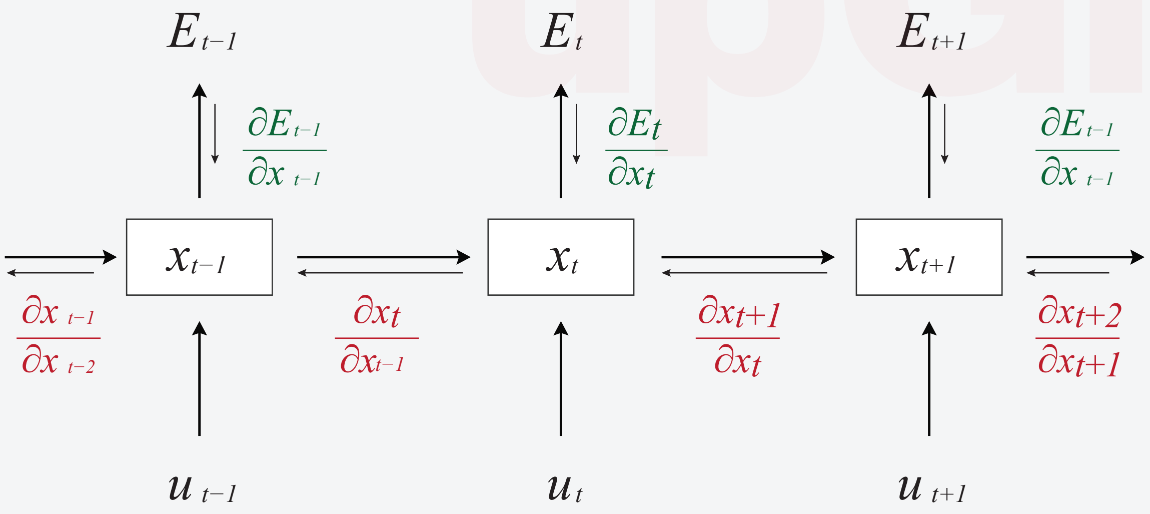
- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
- If the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
- LSTM does provide an easier way for the model to learn long-distance dependencies

BPTT

How does backpropagation work in RNNs?

We have a sequence of inputs x(t), and we will compute a sequence of hidden states h(t) and cell states c(t). On timestep t:

- RNNs, in this setting, are often trained using truncated backpropagation through time (BPTT).
- Forward pass: Step through the next k1 time steps, computing the input, hidden, and output states.
- Compute the loss, summed over the previous time steps.
- Backward pass: Compute the gradient of the loss in relation to all parameters, accumulating over the previous k time steps (this
- requires having stored all activations for these time steps).
- Clip gradients to avoid the exploding gradient problem.


E_{t-1}
E_t
E_{t+1}
$\frac{\partial E_{t-1}}{\partial x_{t-1}}$
$\frac{\partial E_t}{\partial x_t}$
$\frac{\partial E_{t-1}}{\partial x_{t-1}}$
x_{t-1}
x_t
x_{t+1}
$\frac{\partial x_{t-1}}{\partial x_{t-2}}$
$\frac{\partial x_t}{\partial x_{t-1}}$
$\frac{\partial x_{t+1}}{\partial x_t}$
$\frac{\partial x_{t+2}}{\partial x_{t+1}}$
u_{t-1}
u_t
u_{t+1}

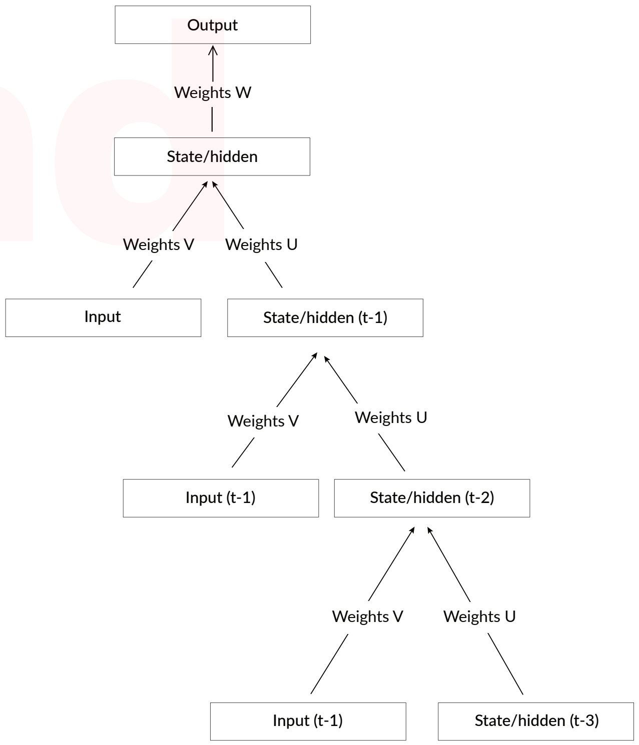


Figure 5: The effect of unfolding a network for BPTT (T = 3).

RNN Challenges

What are the challenges in training RNN?

- Vanishing: Derivatives go to zero - parameters not updated
- Exploding: Derivatives get very large - cause saturation

Gradient Clipping: This is the solution to the exploding gradient issue. Gradient Clipping is a method where the error derivative is changed or clipped to a threshold during backward propagation through the network, and using the clipped gradients to update the weights.

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further. Vanishing: derivatives go to zero - parameters not updated. If the gradients are small Vanishing gradients, learning very slow or stops Solution: introducing memory via LSTM, GRU, etc.

RNN Challenges

LSTM vs GRU

- LSTM (Long Short Term Memory): LSTM has three gates (input, output and forget gate)
- GRU (Gated Recurring Units): GRU has two gates (reset and update gate).
- GRU exposes the complete memory unlike LSTM, so applications that act as an advantage might be helpful.
- GRUs train faster and perform better than LSTMs on less training data

Type of gate	Output gate Io	Output gate Io
Update gate Iu	How much past should matter now?	How much past should matter now?
Relevance gate I,	Drop previous information?	Drop previous information?
Forget gate If	Erase a cell or not?	Erase a cell or not?
Output gate Io	How much to reveal of a cell?	How much to reveal of a cell?

Industry Applications

- One to many RNN - Image Captioning: image -> sequence of words
- Many to One RNN - Sentiment Classification: Sequence of words -> sentiment
- Many to Many - Machine Translation: seq of words -> seq of words
- RNN-LMs speech recognition, machine translation, summarization
- Google Smart Reply is an RNN-applied use case