

RNN

What are RNNs

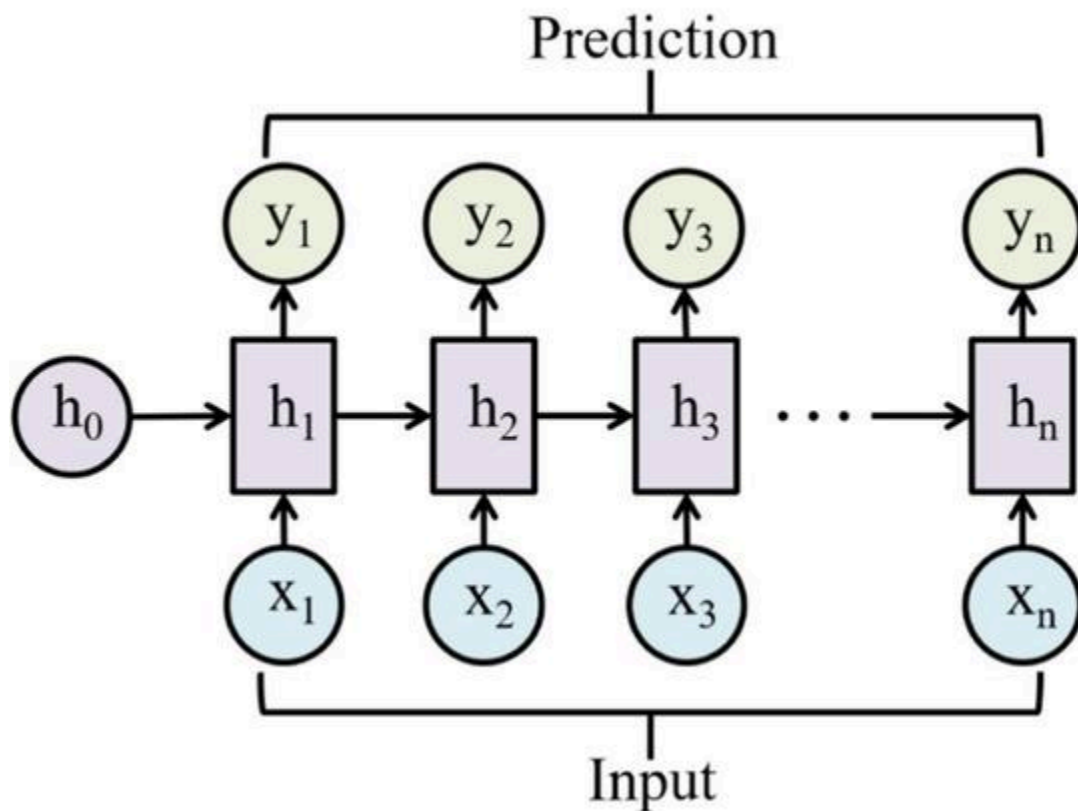
- Short for Recurrent Neural Network
- An update of the neural network that deals with sequential data & thus includes context (short term memory).
- The way that RNNs introduce the short term memory is through the usage of hidden states, hence the name recurrent.
- Hidden states are just a fancy term that represents the storage of the previous state (activation output) of the last sequence.

How RNNs Work

- So suppose we have the following sequence: "The cat sat on the mat":
 - First split the sequence to tokens, for simplicity we will just consider each word as a token, thus you would end with the following list of tokens: ["cat", "sat", "on", "the", "mat"].
 - Assign each token to an ID, since NNs only deal with numeric values: [0, 2, 3, 5, 6].
 - Initialize an embedding to each token, the embedding is just a n-dimensional vector of your size of choice, the greater the dimension the better the semantic capture however the harder the training would be (the choice of the dimension is your call).
 - We will construct our Recurrent Neural Network Architecture, say we are going to have only one layer with a hidden size of 2 (equivalent to the number of neurons in NNs).
 - For each training sample / sequence, pass token by token to that RNN architecture. First you would pass the embedding of the first token to the

neurons, then the final activation function output will be passed along with the embedding of the second token, and so on till all the tokens of the sequence have been passed.

- Hidden State: $h_t = \tanh(W_{xh} * x_t + W_{hh} * h_{t-1} + b)$
- Final output: $y_t = W_{hy} * h_t + b_y$
- Where x_t is the input token embedding and h_{t-1} is the previous hidden state
- Note that for each sequence the weights are shared (they aren't learned, only learned across different sequences)



- Repeat for all the sequences you have.

Types of RNNs

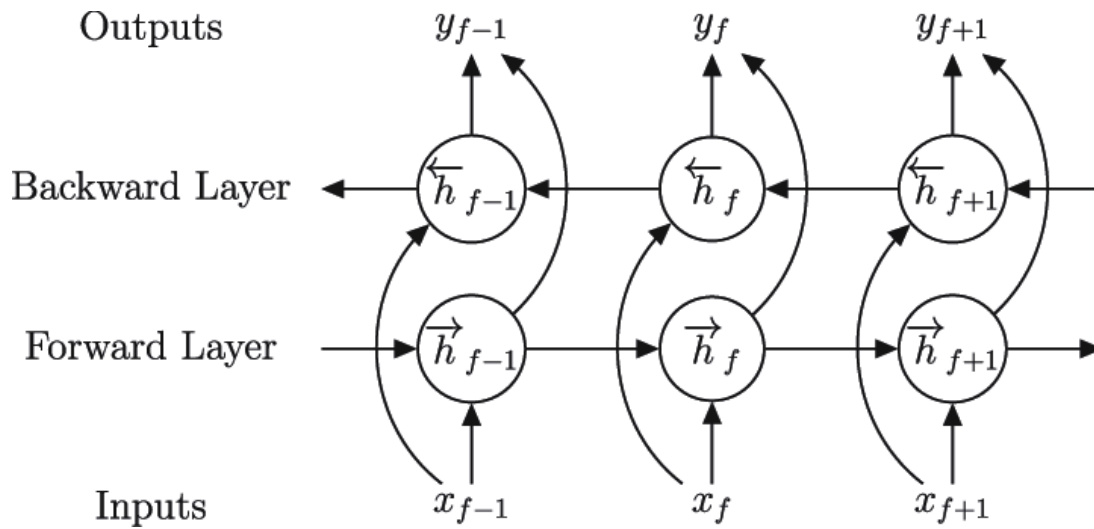
- RNNs have many versions depending on the task:
 - One to many, like in music generation (input is one note, output is a song)
 - Many to many, like in translation (input is sequence in certain language, output is a sequence in another language), NER (input is a sequence, output is the entities)
 - Many to one, like sentiment analysis (input is sequence, output is the sentiment)

Disadvantages of RNNs





- Sequential processing is slow (can't be parallelized, however solved in the transformer architecture).
- Vanishing and exploding gradients can be present, hence can't include long contexts (solved in the LSTM & GRU architectures).
- Tanh and sigmoid activation functions can lead to vanishing gradients.
- Relu activation function can lead to exploding gradients.

Variations of RNNs

- Bidirectional RNN (BRNN), a variation of the RNN that processes the sequence in both paths **forward (past → present)** and **backward (future → present)** — to capture **context from both past and future** tokens.
- This could be beneficial where in some applications the word doesn't depend only on past values but also on future words.
- Works the same as in RNN but introduces an additional Backward pass that processes the sequence in the opposite direction (**Two RNNs** are used).
- For each token the output from the forward and backward pass are concatenated and then that concatenated output is used to form the final output.



Applications of RNNs

	x - input		y - output
Speech recognition		\Rightarrow	"Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair."
Music generation	\emptyset	\Rightarrow	
Sentiment classification	"Decent effort. The plot could have been better."	\Rightarrow	
DNA sequence analysis	ACTGTACCCATGTGACTGCCC	\Rightarrow	ACTGTACCCATGTGACTGCCC
Machine translation	"El que no arriesga, no gana."	\Rightarrow	"If you don't take risks, you cannot win."
Video activity recognition		\Rightarrow	Running
Name entity recognition	"Ygritte says Jon Snow knows nothing."	\Rightarrow	"Ygritte says Jon Snow knows nothing."

Additional Notes

- How Embedding learning works in RNN:

- You start with an embedding matrix:

$$\text{Embedding} \in \mathbb{R}^{(V \times D)}$$

where:

- V: Vocabulary size
- D: Embedding dimension (e.g., 100)
- Every token (word ID) maps to one row in this matrix — its vector representation.
- During training, for every sequence, the model:
 - i. Look up the embeddings for each word.
 - ii. Uses them as inputs to the RNN.
 - iii. The loss (e.g., classification error) is backpropagated through the whole model — including the embedding matrix.
 - iv. The optimizer (e.g., Adam) updates only the rows of the embedding matrix that were used in the batch.
-