Recurrent Neural Networks

A type of sequential model specifically designed to work on sequential data.

→When we work with multiple samples whose order matters, we call that a sequence.

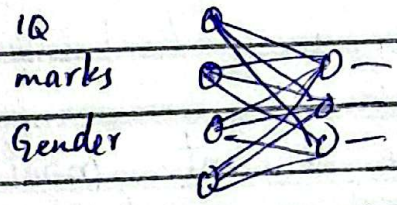
* ANN work well with tabular data.
* CNN work well with Images.

Reasons to not use ANN for Sequence Data:

1) Sequence data is of any length (ANN work with fixed input)
   - text input — varying size
   - If we apply zero padding (It would be unnecessary computation)
   - Still if user input higher length of input it would be problem.

2) Sequence contains some meaning
   - In ANN, all data input at once.
   - It totally disregarding the sequence info.

IQ, marks, gender | Placement



- RNNs are a special class of neural network which have memory feature, which remember past inputs.
- RNN comes in 1986 but famous recently.
- RNN are usually thought of as a stepping stone to understanding fancier things like LSTM and Transformer.
- Just like other neural networks, RNN have weights, biases, layers and activation functions.

→ The big difference is that RNN also have a feedback loops.
→ To deal with the problem of variable input size is to use RNN.

Types of Neural Networks (by structure)

1. Feedforward Neural Networks:
   → Information flows in one direction. (input → output). No loops.
   → Feed forward Neural Networks just focus on current state.

Examples:
- MLP – Basic fully connected network
- CNN – Used for images & Spatial data
- Autoencoders – For data compression and reconstruction.
- GANs (Generator) – Often built with FNN or CNN
- Transformers – Use feedforward layers with self-attention (not recurrent)

2. Recurrent Neural Networks:
   → It consider both the current input and past information by maitaining a hidden state that is updated over time.
   → Previous outputs influence current processing. Designed for sequences.

Examples:
- Basic RNN
- LSTM – Handles long-term dependencies.
- GRU – Simpler alternative to LSTM.

How to feed data to RNN (Transforming textual data)

We input data to RNN in the below format.
(time steps, input-feature)

Reviews	Sentiments
movie was good	1
movie was bad	0
movie was not good	0

→ The first step would be to convert text into vectors/numbers.
→ Inputs are in English, our ML/DL algorithms did not understand it, we need to convert it into numbers/vectors.
→ Simple way is Integer Encoding.

1) Integer Encoding
   a) Form vocabulary:
      - Find how many unique words document have.
      - Give an integer value to every unique word.
   b) Replaces sentences with its integer values.
   c) Padding
      - Bring all to same size.

→ size of each review is different.

vocab → 5 words

[10000] [01000] [00100] [00010] [00001]

review 1: (3, 5)
review 2: (3, 5)
review 3: (4, 5)

2. Embeddings
→ Convert every word into a vector, on the basis of the whole document.
→ The vector here we got is dense not sparse.
→ Dense: Values are non-zeros.
→ Sparse: Most of values are 0s.

this is nice
↓
[0.7 0.1 0.3]

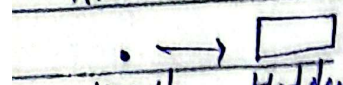
Benefits:
- Dense representation
- Semantic Meaning (word use in which context)

→ word2vec and glove are word embedding techniques

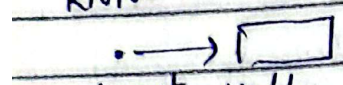
RNN Architecture:

→RNN is very similar to ANN

ANN:



RNN:



	Review	Sentiment
x_1	movie (x_{11}) was (x_{12}) good (x_{13})	1
x_2	move (x_{21}) was (x_{22}) bad (x_{23})	0
x_3	movie (x_{31}) was (x_{32}) not (x_{33}) good (x_{34})	0

→We represent every word as vector.

[[0000] [0|000] [00|00] [000|0] [0000|]

→But there two major differences.

1st:

How you fed data in your network?

→In RNN input is sent on the basis of time.

time basis

t = 1 x_{11} →

t = 2 x_{12} →

t = 3 x_{13} →

→So first major difference is that complete input is not fed up at once in RNN.

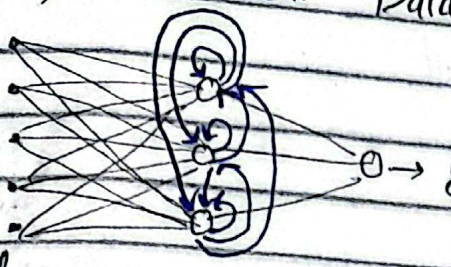
2nd:

→ANNs are feed forward neural networks. In forward moves from input to output.

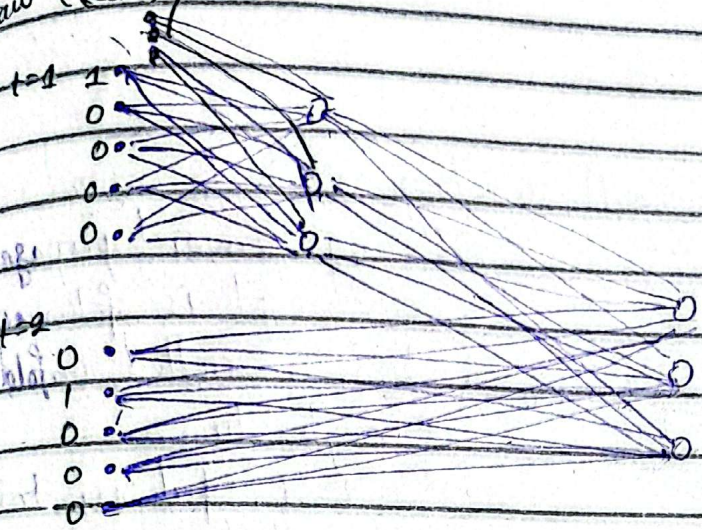
→RNNs are not feed forward. In RNN there is a concept of state.

→RNNs make use of an idea called state. This is just a description of a system (such as neural network) at any given time.

Architecture for Current Dataset:



Draw Clearly:



→ We are working with t=1 [10000]

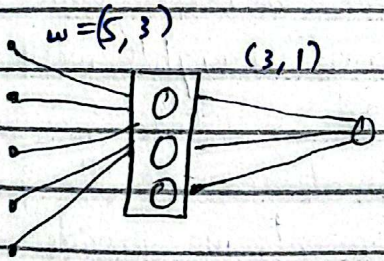
→ It will go to hidden layer, some calculations done, activation will applied. Every node has an output, which will be sending back to the nodes but for t=2.

→ For t=2 [01000], sent to the same node but this time, node output come as well. It will continue.

→ Now you may be thinking that for t=2 we got feedback from first, but for t=1 what will we get.

→ For t=1, feedback will be 0 or random values

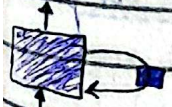
Number of Weights & Biases:



$w_1=15$ $w_2=3\times3$ $w_3=3$

Weights = 27

Biases = 4



Put-in-vanilla
Remove-1-scoop
Add-sauce
Add-cherry

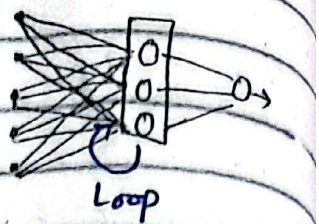
→ The loop on the right represent the state between one input and the next

→ After each input, the system (represented by the big, light blue box) creates a new state, which goes into the black square.

→ This square is called the delay and we can think of it as a little piece of memory.

RNN Forward Propagation:

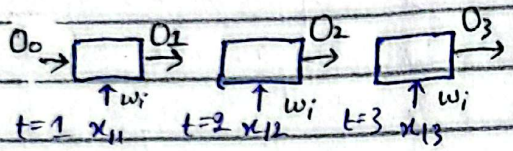
Review	Sentiment
x_{11} x_{12} x_{13}	1
x_{21} x_{22} x_{23}	0
x_{31} x_{32} x_{33}	0



x_{11} → vector

[1 0 0 0 0]

↳ One by one we will sending words to neural network



1) $f(x_{11}w_i + O_0w_h + b_1)$

→ This is done inside the above box.

→ O_1 will be output from this node.

$O_1 \rightarrow (1,3)$

$w_h \rightarrow (3,3)$

2) $f(x_{12}w_i + O_1w_h + b)$

$(1,5)(5,3) + (1,3)(3,3)$

$(1,3) + (1,3)$

$O_2 \rightarrow (1,3)$

3) $f(x_{13}w_i + O_2w_h + b)$

$O_3 \rightarrow (1,3)$

This is how forward propagation in RNN works.

(To keep consistency we fed O_0 which is basically (1,3) vector of 0's or random)

→ During forward propagation a concept is followed which is called unfolding through time.

→ On t=1, fed x_{11} take dot product with weight

→ Apply activation function.

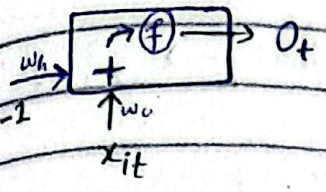
→ At t=2, we again use same network with the same weights, we'll fed 2nd word.

→ This time one thing is different, we provide an extra input to network which is O_1

→ After t=3, our all words finished, take dot product of O_3 with three weights we get (1, 1) which is scalar.

→ Apply Sigmoid on scalar, you will get output.

Summary



i = row number

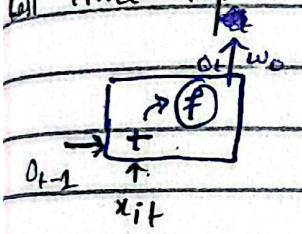
t = time step

x_{11} →

$f(x_{it} w_i + O_{t-1} w_h)$

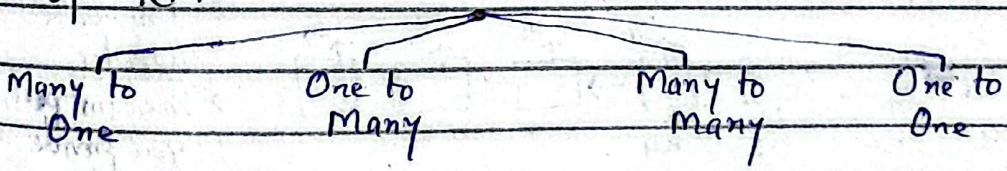
→ Imagine RNN as a box, where two input goes.
x_{it} and O_{t-1} (output of previous time step)
→ x_{it} goes through a weight matrix called w_i and O_{t-1} goes through a weight matrix w_h.
→ First perform dot products of input and weights, previous output and weights, then sum and then pass the vector to an activation function. You'll get current timestep output O_t.
→ If this is last timestep then pass it with w_o.

Last Time Step:



$\hat{y} = g(O_t w_o)$

Types of RNN



Many to One:

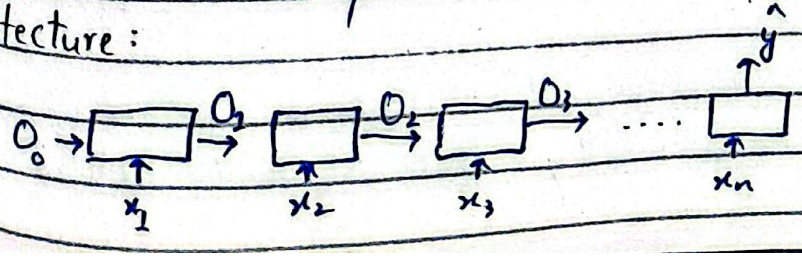
input : sequence (sentence, time series)

output : non sequence (number → scalar (0, 1))

e.g:

Sentiment Analysis

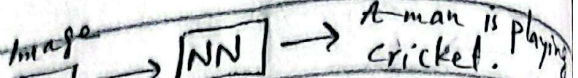
Architecture:



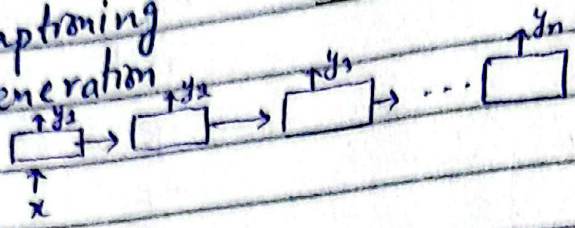
2) One to Many (Image ...)

input : non sequence
output : Sequence

e.g: Image Captioning
Music Generation



Architecture:



3) Many to Many

input : Sequential Data
output : Sequential Data

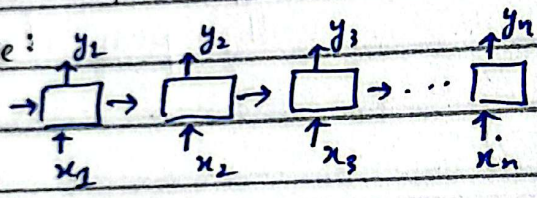
It is also called Seq2Seq

- Same Length: Same length
number of input = number of output

e.g: POS Tagging: Variable length

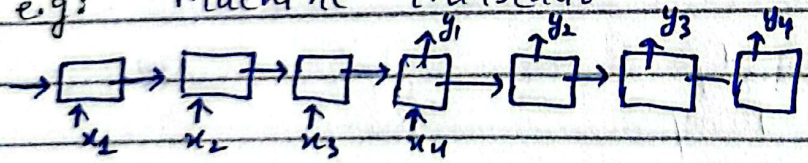
My name is Imran
↓ ↓ ↓ ↓
Pronoun noun verb Noun

Architecture:



- Variable Length:

e.g: Machine translation

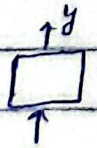


* Such architecture is also called encoder decoder.
* The first part, where you provide input is called encoder.
* The part where you get output called decoder.

4) One to One

input : Non Sequential data
output : Non Sequential data

e.g: Image Classification



 → 1, 0

Basically, there is no recurrence.
Technically, this is not RNN.

7 July 2025

Backpropagation in RNN

RNN is a neural network. Neural networks do learning. Learning is done through backpropagation. Here it is called BPTT.

dataset

text	label
cat mat rat	1
rat rat rat	1
mat mat cat	0

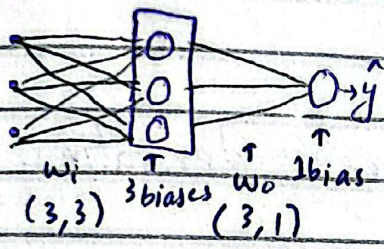
→

x	y
[1 0 0] [0 1 0] [0 0 1]	1
[0 0 1] [0 0 1] [0 1 0]	1
[0 1 0] [0 1 0] [1 0 0]	0

↳ First convert text into vectors.

↳ Vocab: cat mat rat

[1 0 0] [0 1 0] [0 0 1]

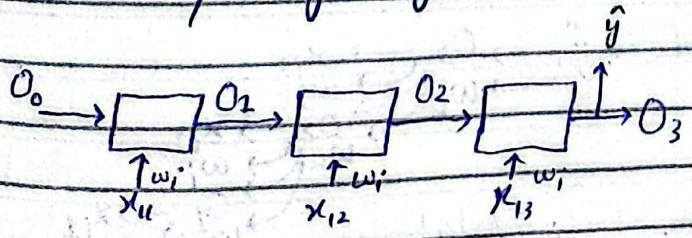


Forward Propagation:

↳ Reviews will be inserted word by word.

- First send [1 0 0], after calculations you get O_1.
- Then, send [0 1 0], after processing you get O_2.
- Then, input [0 0 1], after processing you'll get O_3.

↳ Since no words remaining in input, you move forward and you get $\hat{y}$.



$O_1 = f(x_{11} w_i + O_0 w_h)$

$O_2 = f(x_{12} w_i + O_1 w_h)$

$O_3 = f(x_{13} w_i + O_2 w_h)$

→ When we get O_3, we'll use it to get $\hat{y}$.

$\hat{y} = G(O_3 w_o)$

→ Now we got $\hat{y}$, we also y. Use them to calculate loss.

$$L = -y \log \hat{y}_i - (1-y_i) \log (1-\hat{y})$$

→ After calculating loss, our goal is to minimize loss. The way to minimize loss is gradient descent algorithm. (optimization algorithm)
→ what we have to do is to find the values for parameters which reduce loss.

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} L$$

$$w_h = w_h - \alpha \frac{\partial}{\partial w_h} L$$

$$w_0 = w_0 - \alpha \frac{\partial}{\partial w_0} L$$

→ Now, calculate these three derivatives.

1. $\frac{\partial L}{\partial w_0}$

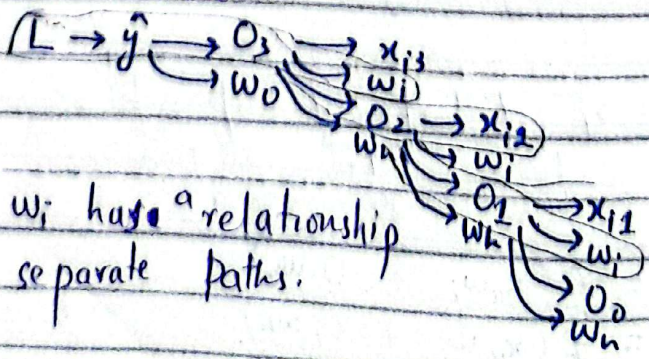
→ This one is closer to $\hat{y}$, so simple to calculate

$L \rightarrow \hat{y} \longrightarrow O_3$
$\hat{y} \rightarrow w_0$

→ Change in loss depends on $\hat{y}$, because y is constant.
→ $\hat{y}$ is calculated as $\hat{y} = \sigma(O_3 w_0)$
→ We can write using chain rule.

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_0}$$

2. $\frac{\partial L}{\partial w_i}$



→ L and w_i have a relationship of three separate paths.

→ There are three paths, so we have to calculate differentiation accordingly.

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_i}$$

→ Here we have three terms because our RNN unfold in time, because our dataset have 3 words.

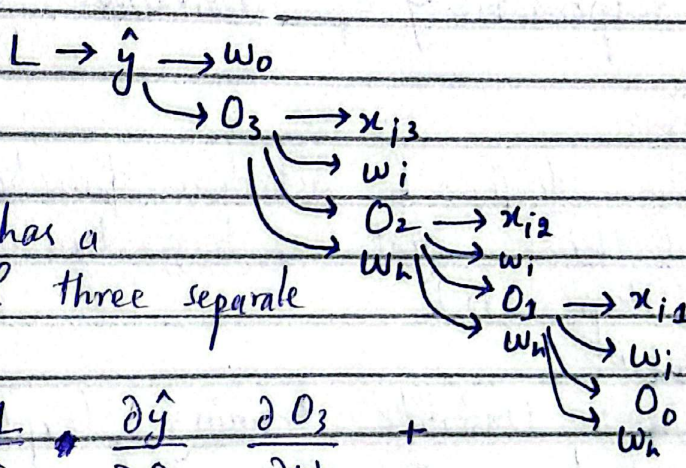
Compact form:

$$\Rightarrow \frac{\partial L}{\partial w_i} = \sum_{j=1}^{t=3} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_i}$$

$$\Rightarrow \frac{\partial L}{\partial w_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_i}$$

n = time steps

3. $\frac{\partial L}{\partial w_h}$



→ L and w_h has a relationship of three separate paths.

$$\frac{\partial L}{\partial w_h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_h} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_h} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_h}$$

Compact form:

$$\frac{\partial L}{\partial w_h} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_h}$$

n = time steps

→ As these three derivatives calculates, apply gradient descent step.

→ First it work with some initial weights for w_i, w_h, w_o

Problems with RNN:

RNN ⟶ Sequential Data (text, time series)

↳ But RNN not used too much.

↳ It suffers from two major problems.

→ The more we unroll a recurrent neural network, the harder it is to train.

1. Problem of long-term dependency.

→ This problem arise because of vanishing gradient problem.

→ When a gradient becomes very small, learning slows down, and if a gradient becomes zero, learning stops entirely.

→ The more the timesteps are, the larger the term will be. So your long-tem dependency will get small. The responsibility of derivatives comes on short term.

Solutions:

1) Use different activation functions
2) Better weight initialization.
3) Skip RNNs
4) LSTM

2. Problem 2: Unstable training (Exploding Gradient)

→ Gradients gets larger every time we step backward through the unrolled diagram.

Solutions:

1) Gradient Clipping
2) Controlled learning rate
3) LSTM

16