RNN (Recurrent Neural Networks)

Objective of RNN:

→ RNN should be used for the applications which has variable length inputs.

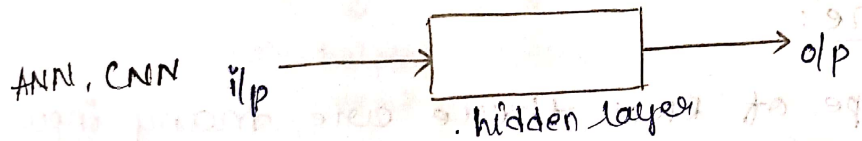
→ when the input features has dependency then RNN is most suitable Neural network to use.

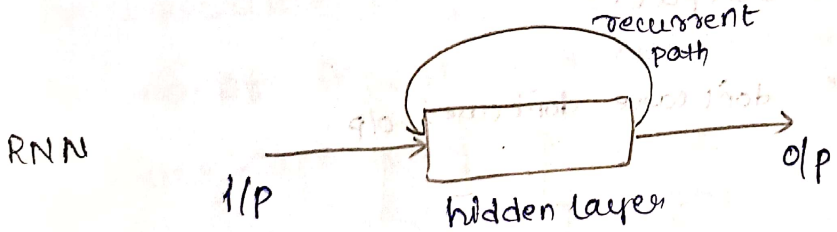
Note:

CNN and feed-forward NN's can only be used when:

1. Input size is fixed
2. There is no dependencies among the inputs

→ RNN has a special feature called memory cell which remembers the past information whereas CNN and ANN doesn't have.





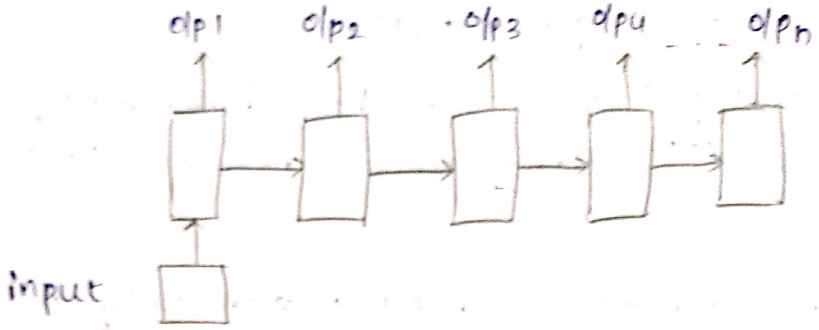
Types of RNN:

There are 4 types of RNN's

1. one to one
2. one to many
3. many to one
4. many to many (or) sequence to sequence

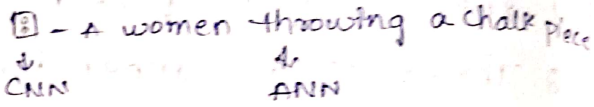
1. One to many.

One to many RNN has one input and many outputs.



Ex: Image captioning (one image - variable length text)

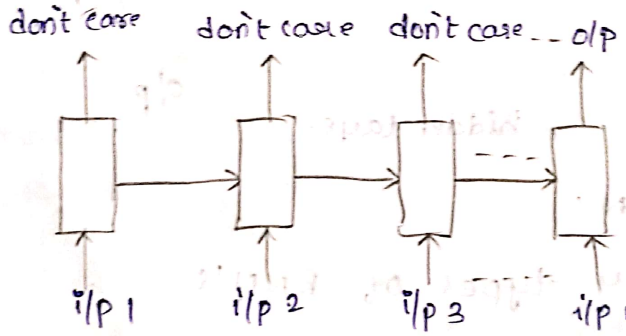
Encoder - CNN
Decoder - RNN



→ RNN is a neural network which is used to process sequence data (or) time-series data.

2. many to one:

In this type of RNN there are many inputs but only one output.

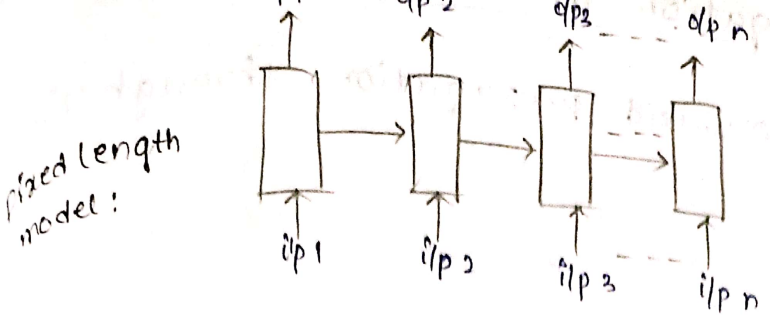


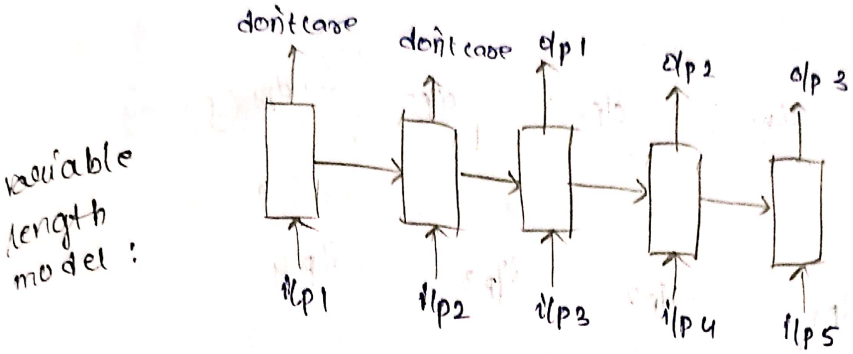
Ex: Sentiment Analysis.
Forecasting stock price

3. Many to many:

Many to many RNN model is also called as sequence sequence model.

There are 2 types of Sequence2Sequence models.





(i) fixed length sequence 2 sequence

(ii) variable length sequence 2 sequence

→ In fixed length sequence model for every input there will be exactly one output.

Ex: Srivani is laughing : Parts of speech (POS) tagging

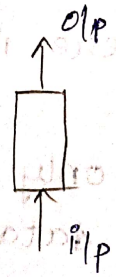
Srivani → Noun, is → helping verb, laughing → verb

→ In variable length sequence 2 sequence model the no. of inputs and outputs will vary.

Ex: video captioning

4. One to one :

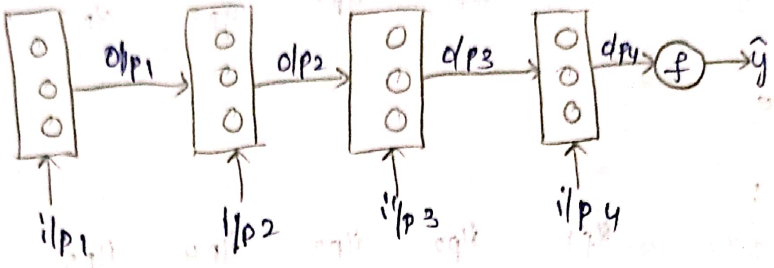
one to one RNN has one input and one output only.



Ex: Image Cap classification

Forward Propagation in RNN

RNN follows forward propagation through time approach



→ the input shape of SimpleRNN is represented as

(no. of timesteps, no. of features)

Note: In ANN, the entire input is fed into the network at once

→ In RNN, the input is fed into the network at time intervals (called as time steps)

Ex:

Let us consider an example Sentiment Analysis on the dataset as follows:

Feature	Label
movie was good	1
movie was bad	0
movie was not good	0

In the above dataset there is one feature and one label.

Any NN can understand only numbers but our dataset contains textual data. we need to convert this textual data into numeric form by using any encoding techniques like one-hot encoding.

The given corpus contains 5 vocabulary words

movie, was, good, bad, not

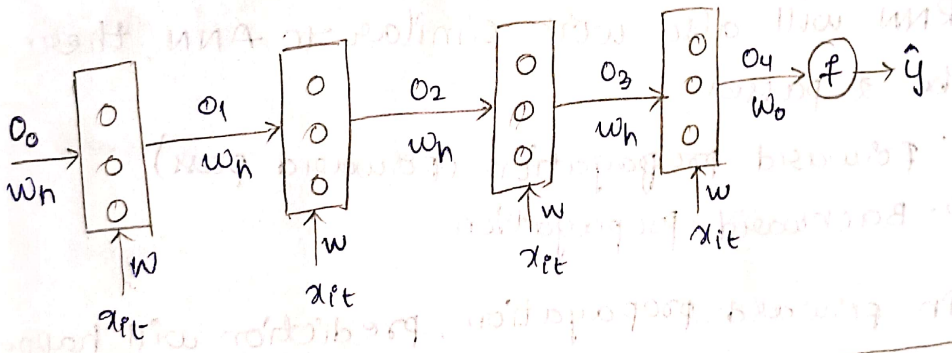
vocabulary	one-hot vector
movie	[1 0 0 0 0]
was	[0 1 0 0 0]
good	[0 0 1 0 0]
bad	[0 0 0 1 0]
not	[0 0 0 0 1]

→ The input shape for this dataset is (4, 5)

(4: longest review length, 5: vector size)

→ when we built a simple RNN for this data

	x_{11}	x_{12}	x_{13}	
x_1	movie	was	good	
x_2	x_{21} movie	x_{22} was	x_{23} bad	
x_3	x_{31} movie	x_{32} was	x_{33} not	good



$o_1 = f(x_{it} w + o_0 w_h + b)$

$o_2 = f(x_{it} w + o_1 w_h + b)$

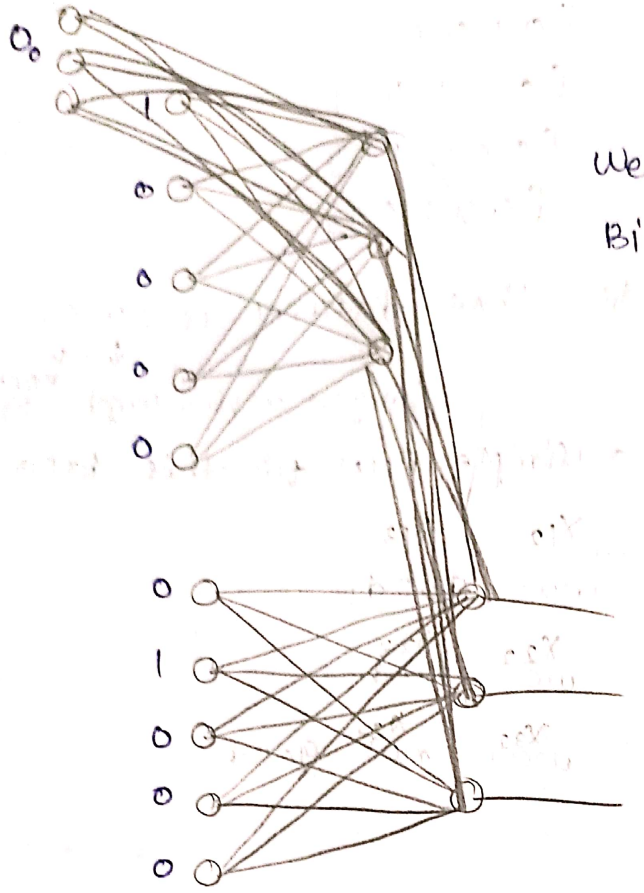
$o_3 = f(x_{it} w + o_2 w_h + b)$

$o_4 = f(x_{it} w + o_3 w_h + b)$

$o_t = f(x_{it} w + o_{t-1} w_h + b)$

→ the activation function usually used in RNN is tanh()

Forward propagation through Time:



Weights = 9+9+15 = 33

Biases = 3

→ RNN will also work similar to ANN, there will be 2 passes

1. forward propagation (forward pass)
2. Backward propagation

1. In forward propagation, prediction will happen we will calculate $\hat{y}$.

→ After calculating $\hat{y}$ we will find loss by using some loss function

→ If you want to minimize the loss we will perform backward propagation by using Gradient Descent Algorithm.

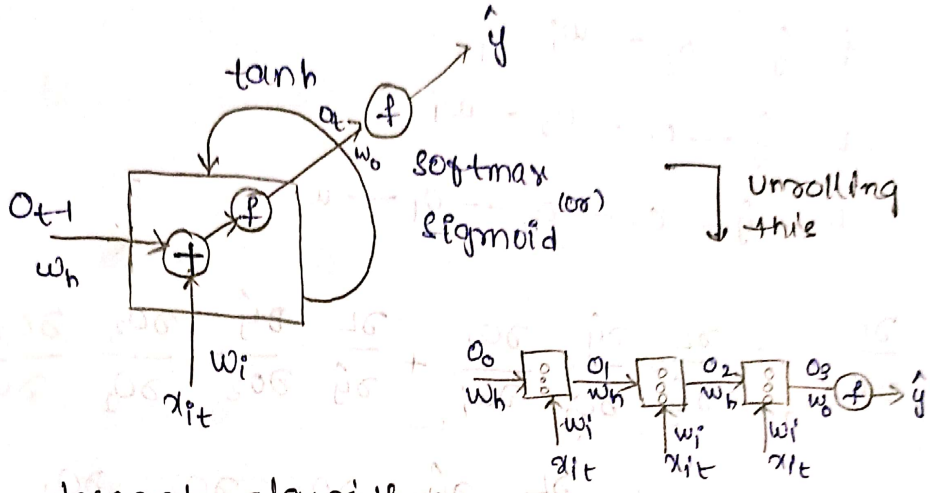
→ The formula for GD Algorithm to change the weights and biases are

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

will happen.

→ training a neural network means optimizing the weights and biases



Gradient descent algorithm:

To calculate the weights and biases in backward propagation the formulae are:

$$w_{inew} = w_{iold} - \eta \frac{\partial L}{\partial w_{iold}}$$

$$w_{hnew} = w_{hold} - \eta \frac{\partial L}{\partial w_{hold}}$$

$$w_{onew} = w_{oold} - \eta \frac{\partial L}{\partial w_{oold}}$$

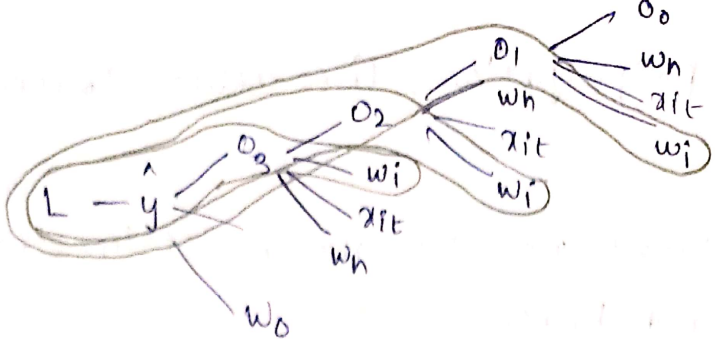
$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

The above are the 4 learnable parameters in RNN.

(1) $$w_{onew} = w_{oold} - \eta \frac{\partial L}{\partial w_{oold}}$$

$$= w_{oold} - \eta \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_o} \right)$$

(2) $$w_{inew} = w_{iold} - \eta \frac{\partial L}{\partial w_{iold}}$$



$L - \hat{y} - o_3 - w_i$

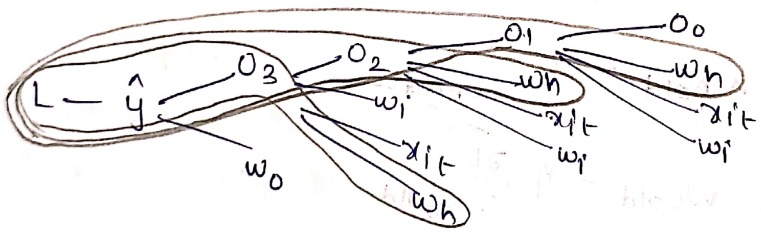
$L - \hat{y} - o_3 - o_2 - w_i$

$L - \hat{y} - o_3 - o_2 - o_1 - w_i$

$$\frac{\partial L}{\partial w_{i_{old}}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_i} +$$

$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_i}$$

(3) $w_{h_{new}} = w_{h_{old}} - \eta \frac{\partial L}{\partial w_{h_{old}}}$



$L - \hat{y} - o_3 - w_h$

$L - \hat{y} - o_3 - o_2 - w_h$

$L - \hat{y} - o_3 - o_2 - o_1 - w_h$

$$\frac{\partial L}{\partial w_{h_{old}}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial w_h} + \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_h} +$$

$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_h}$$

RNN suffers with a problem called vanishing / exploding gradients.

vanishing gradients means, while back propagation the gradients (weights) becomes very small probably zero. which leads to dead neural network

Exploding gradients means, while back propagation the gradients (weights) becomes very large probably infinity.

To overcome the above problem we use RNN variants

1. LSTM (Long Short Term memory)
2. GRU (Gated Recurrent Unit)

these two neural networks uses gates concept internally to overcome vanishing/exploding gradients problem.

Deep Neural Network Libraries

1. Keras - Google
2. pyTorch - Facebook
3. Theano - microsoft

PyTorch

PyTorch is used for Implementation of Image classification with CNN.

Steps to perform Image classification with CNN using PyTorch:

1. Load the Data into objects
2. Split the data (train data, validation data)
3. Construct the neural network
4. Construct the base model
5. fit, Evaluate & predict

→ In pyTorch, the major libraries are Torch and torchvision.

→ Torchvision library is majorly used to implement computer vision problems

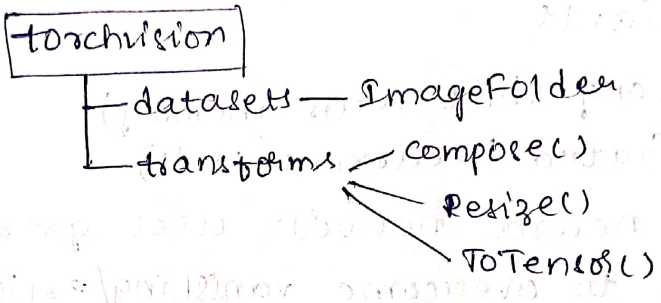
1. Loading the data into objects

import torchvision package

from torchvision. import transforms module

from torchvision. datasets import ImageFolder

datasets: subpackage, ImageFolder: class



object

train_data = ImageFolder(train, transforms.compose([transforms.Resize((32,32)), transforms.ToTensor()]))

object

test_data = ImageFolder(test, transforms.compose([transforms.Resize((32,32)), transforms.ToTensor()]))

ImageFolder(p1, p2)

↓ ↓

directory. Preprocess the image

Resize() ToTensor()

2. Splitting the data

from torch.utils.data import random_split

train_ds, valid_ds = random_split(train_data, lengths=[0.8,0.2])

random_split(p1, p2)

↓ ↓

dataset %

Batch loading data into batches

```
from torch.utils import DataLoader
train_dl = DataLoader(train_ds, 128, shuffle = True)
valid_dl = DataLoader(valids, 128, shuffle = False)
#DataLoader(dataset, batch_size, shuffle)
```

Construct the Neural Network

NN
- Conv2D()
- Relu()
- MaxPool2d()
- Linear()

Feature map size:

$$\frac{n+2p-f+1}{s}$$

p = padding size

```
class CNN():
    def __init__(self):
        self.model = nn.Sequential(
nn.Conv2d(3, 32, kernel_size=(3,3), stride=1, padding=1)
nn.Relu()        # in-channel, out-channel
nn.Conv2d(32, 64, kernel_size=(3,3), stride=1, padding=1)
nn.Relu()
nn.MaxPool2d(2,2)
nn.Conv2d(64, 128, kernel_size=(3,3), stride=1, padding=1)
nn.Relu()
nn.Conv2d(128, 128, kernel_size=(3,3), stride=1, padding=1)
nn.Relu()
nn.MaxPool2d(2,2)
nn.Conv2d(128, 256, kernel_size=(3,3), stride=1, padding=1)
nn.Relu()
nn.Conv2d(256, 256, kernel_size=(3,3), stride=1, padding=1)
nn.Relu()
nn.MaxPool2d(2,2)
nn.Flatten(4096, 1024)
nn.Linear(1024, 512)
nn.Relu()
nn.Linear(512, 10)
```