Recurrent Neural Networks (RNN)

CNN → designed to work on image data.

MLP → designed to work on vector input.

RNN → for sequence data – sequence of words, time series etc.

Ex→ This phone is very fast. ← signifies quality of phone.

The sequence of occurence is important.

* All of the existing featurizations – BoW, W2V, tf-idf etc. discard sequence info.

Some real-life applications –

1. Sentiment analysis using natural language.
2. Stock market prediction.
3. Machine translation.
4. Speech recognition – Alexa, Google Now, Siri.
5. Image captioning. (can be used in image search).

core-idea :→ output depends on the sequence of inputs.

* So we need a new type of NN which retains and leverages sequence info.

Problem with old NN design

* Say we are using 1-hot encoding for words.
* We don't know how many words are there in the sequence.
* Even if we create a large network, keeping in mind worst case, there still exist chances of it being exceeded.

RNN

(e.g.) Amazon food Reviews — Binary classification.

$D = \{x_i, y_i\}_{i=1}^{n}$

x_i → sentence / Seq. of words. y_i → binary.

$x_1 \rightarrow x_{11}, x_{12}, x_{13}, x_{14}, x_{15}$

$x_{ij} \in R^d$ (one-hot encoding).

d = size of vocab.

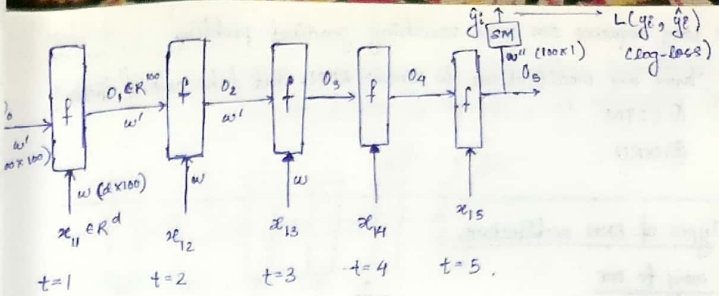


Figure 1

f : sigmoid / tanh / ReLU.

O_0 :– zeros / gaussian random.

① $O_t = f(wx_{it} + w'O_{t-1})$ ② $\hat{y}_i = \sigma(w''O_5)$

3 weights to learn – w, w' & w''.

A typical RNN cell can be represented as

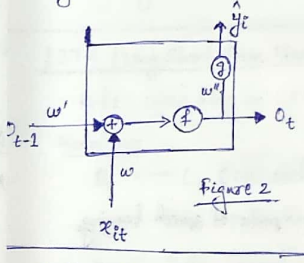


Figure 2

Training RNN : Backpropagation over time

* Forward propagation happens in direction of arrows in fig ①.
* Backward propagation → opposite direction of arrows.
* for the above example, log-loss will do.

BP

$\frac{\partial L}{\partial \hat{y}_i}$ ✓

$\frac{\partial L}{\partial w''} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w''}$ ①, $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_5} \cdot \frac{\partial O_5}{\partial w}$ ②.

$\frac{\partial L}{\partial w'} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_5} \cdot \frac{\partial O_5}{\partial w'}$ ③,

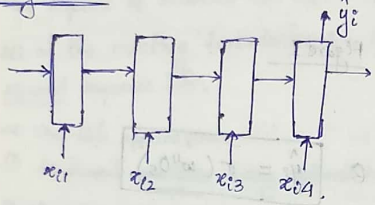
for prev time steps $\frac{\partial L}{\partial w^*} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_5} \cdot \frac{\partial O_5}{\partial O_4} \cdot \frac{\partial O_4}{\partial w}$.

And so on.

* Long sequence can cause vanishing gradient problems.
There are modifications to simple RNN. that take care of these.
① LSTM
② GRU

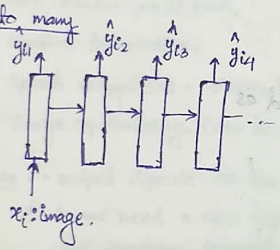
Types of RNN architecture.

① many to one



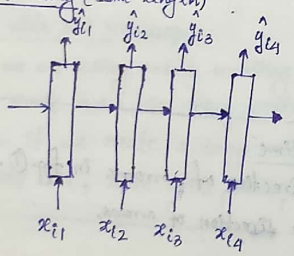
ex → sentiment classification,
→ movie review → rating.

② one to many



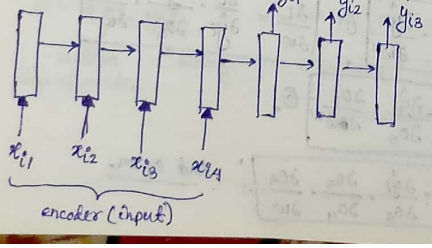
ex → image captioning.

③ many to many (same length)



ex → parts of speech tagging.

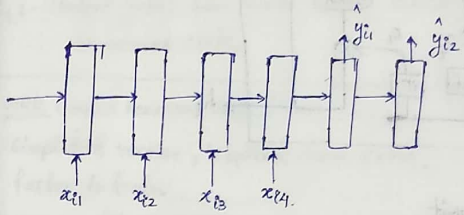
④ many to many (different length)



encoder (input)

Need for LSTM/GRU

Problem with simple RNN. → can't take care of long term dependencies, i.e. when a later o/p depends a lot on earlier i/p.



consider language translation example.

It may so happen that $\hat{y}_{i2}$ depends more on x_{i1} & less on x_{i4}.

* But because of the way RNN cell is designed, due to the depth in time axis, $\hat{y}_{i2}$ depends more on x_{i4} and less on x_{i1}.

* Even in back-prop, the gradient vanishes significantly before reaching x_{i1}.

LSTM (Long Short Term Memory RNN) (1997)

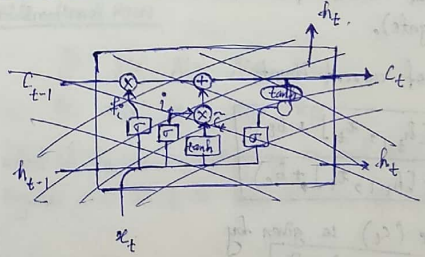
Ref:- colab blog on LSTM.

Notations

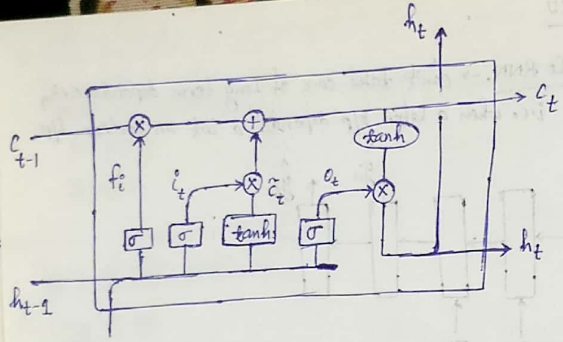
$\theta_t \longrightarrow C_t$ (cell state)

$y_t \longrightarrow h_t$ (output)

$x_{it} \longrightarrow x_t$ (input).



see next page.



LSTM unit

* Long term dependencies could be formed with the help of identity relation (that could be formed using these).

If $f_t = \langle 1, 1, 1, \cdots 1 \rangle$ then $C_t = C_{t-1}$.

& $i_t \times \tilde{C}_t = \langle 0, 0, 0 \cdots 0 \rangle$

* So, just like ResNets here also we can have identity rel^n. when needed (when long term dependencies are more reqd than short term in o/p).

f_t:- (forget gate) decides how much of the prev. cell state should be retained in current state.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f).$$

i_t:- how much of the current input impacts the current state. (input gate).

$\tilde{C}_t$:- featurization of current input.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$C_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

finally, current state (C_t) is given by

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

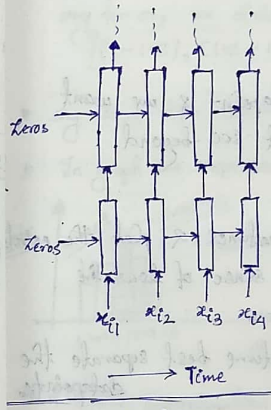
o_t:- (output gate) how much output should be generated based on current state.

GRU (Gated Recurrent Unit). (2014).

* Simplified version, inspired from LSTM.
* Faster to train.
* As powerful as LSTM.

Ref:- slideshare.net / - Recent Progress in RNN and NLP.

Deep RNNs



forward prop → flow in direction of arrows.

backward prop → flow opposite to arrows.

* Exact structure of deep RNN is problem specific.
* All concepts like dropouts, batch Norm etc are applicable.

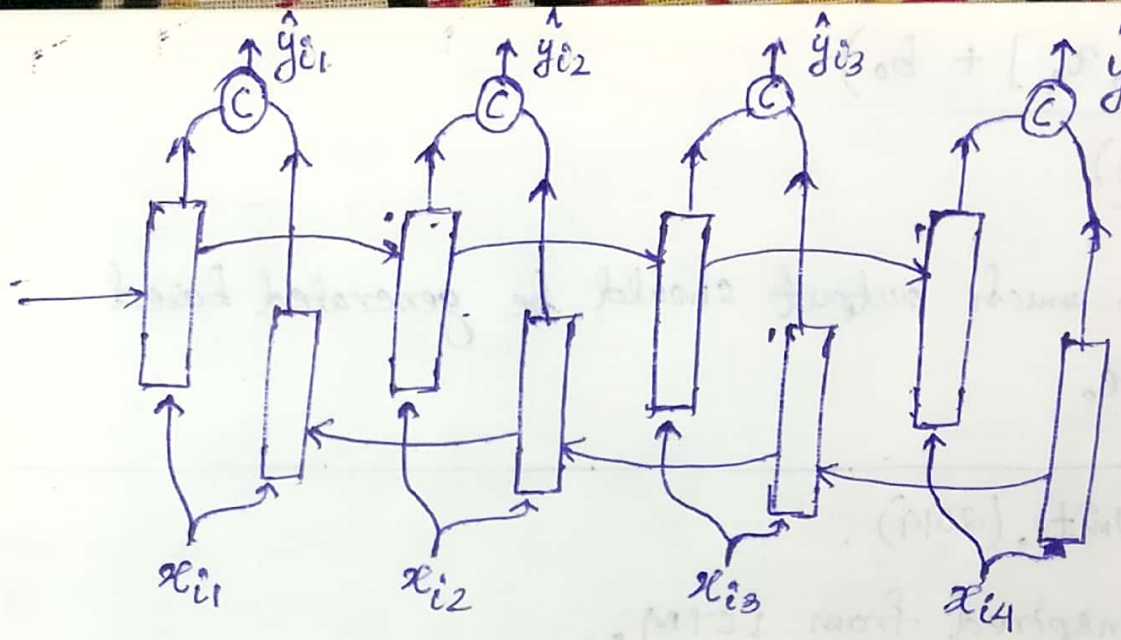
Bidirectional RNN

y_{i1} y_{i2} y_{i3} y_{i4} y_{i5}

x_{i1} x_{i2} x_{i3} x_{i4} x_{i5}.

Q) what if y_{i2} is dependent on x_{i1} and x_{i5}?

→ Regular RNN won't be able to help, as we haven't yet seen x_{i5}.



⟶ Time.

Ⓒ - concatenate

Bidirectional RNN