Sequence Models

Jawwad Shamsi

April 1, 2024

1 Sequence Modelling

Earlier, we have studied Convolutional Neural Networks, CNNs, which are the optimal choice for working the images, or multi-dimensional data. The architectures, we have studies, so far, deals with the multi-dimensional data, where attributes are largely independent of each other. However, in many applications, such as text, speech, and time-series, the data may contain sequential dependencies.[OCW] They are used for many applications such as, image captioning, machine translation, video summarization, DNA analysis, speech recognition, fake news detection, which require sequential processing. Another key application area is the natural language processing (NLP) or Chatboats.

For instance, for figure 1, there could be many possibilities to predict the next step of the ball movement

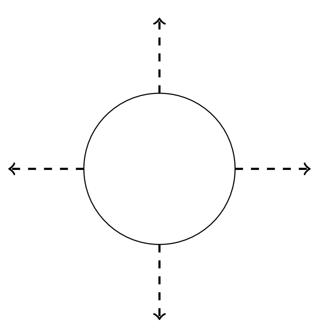


Figure 1: Circle- without sequencing information

However, for figure 2, it is easy to predict the next movement of the ball.

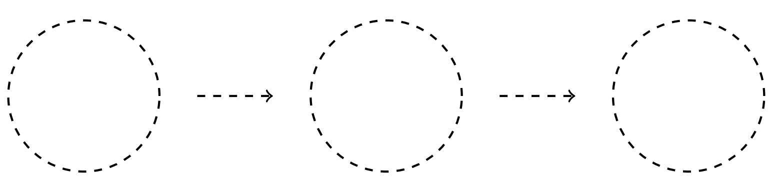


Figure 2: Circle- with sequencing information

Sequential data is around us. Audio can be split up into a sequence of sound waves and text can be split up into a sequence of characters

1.1 Examples of Sequence Learning

Type	Example
One to One	Simple Classification Example
One to Many	Image Captioning
Many to One	Sentiment Analysis
Many to Many	Machine Translation

1.2 Understanding Sequential Problems

Sample Sentence: This morning I went for a walk

Given Sentence: This morning I went for

Predicted Sentence: walk

Given seen words, we have to predict the output.

Number of seen words will, or the length of sentence will vary.

Question Is it a classification problem? Can we use Feed forward neural network?

Numerical Representation How to represent data numerically. Using One hot encoding feature vector.

1. fixed window: Input vector is of fixed length. For instance given these two words "for a", we would like to predict "walk". This has limitations: **Limitations** Cannot model long term dependency because the size is limited e.g. "Peshawar is where I grew up, but I now live in Karachi, I speak fluent —-" We may need long term sentences for prediction. Past 2 words ? 5 words? not enough. need start to finish and as a fixed length vector

2. Bag of Words: A fixed length vector. Each value in the vector represents the count of the no. of words in the sentence. **Limitations** Using just the count is not enough. we will need sequencing information. "The food was good, not bad at all" vs The food was bad, not good at all. We need to capture semantic information.

3. . Use a really big window **Limitations** Things we learn about the sequence won't transfer if they appear elsewhere in the sequence.

4. . Incorporate long term dependency.

1.3 Design Criteria for Sequence Modeling

1. Handle Variable Length

2. Track long-term dependencies

3. Maintain information about order Share parameters across the sequence.

1.4 Recurrent Neural Network

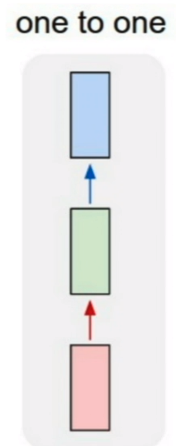


Figure 3: One to One sequence

Data has a flow from input to output. Such a network is limited

Many to many and many to one are also possible

Standard Neural network is limited. Does not retain information about previous output or sequence. Figure 4 shows a block diagram of RNN. Input to the system is x_t, $\hat{y}$ is the final output, and h_t is the intermediate output.

Apply a recurrence relation at every time step to process a sequence.

$$h_t = f_w(h_{t-1}, x_t) \tag{1}$$

At each time step, the function f_w is applied to update the state h_t. where,
h_t = cell state,
f_w = function parameterized by W,
h_{t-1} = output of the previous state t-1, and
x_t = input at the current time step t

Same function and same set of parameters are used at every time step. These are learned during training. Weights are updated during each iteration.

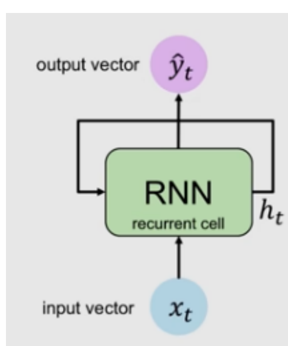


Figure 4: RNN

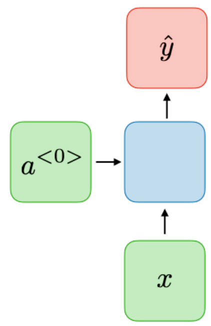


Figure 5: One to One

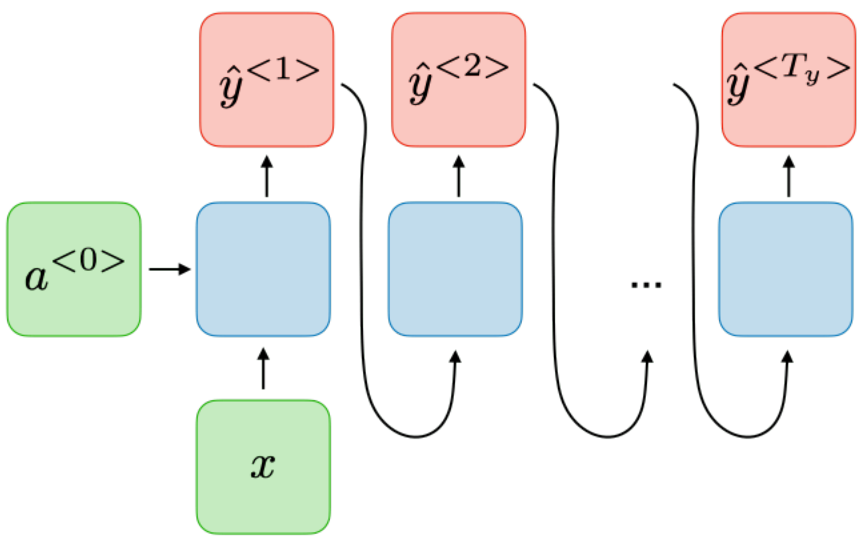


Figure 6: One to Many

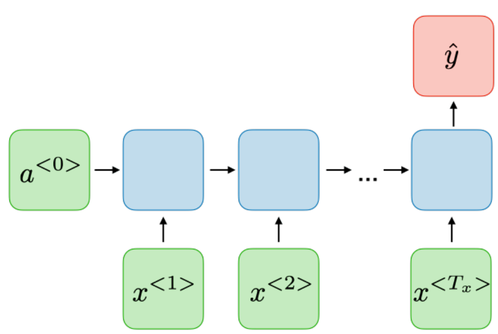


Figure 7: Many to One

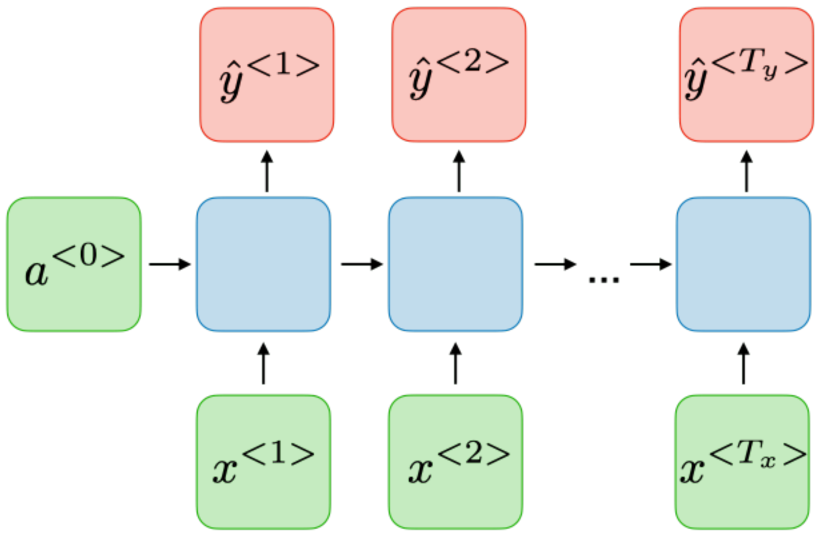


Figure 8: Many to Many, x=y

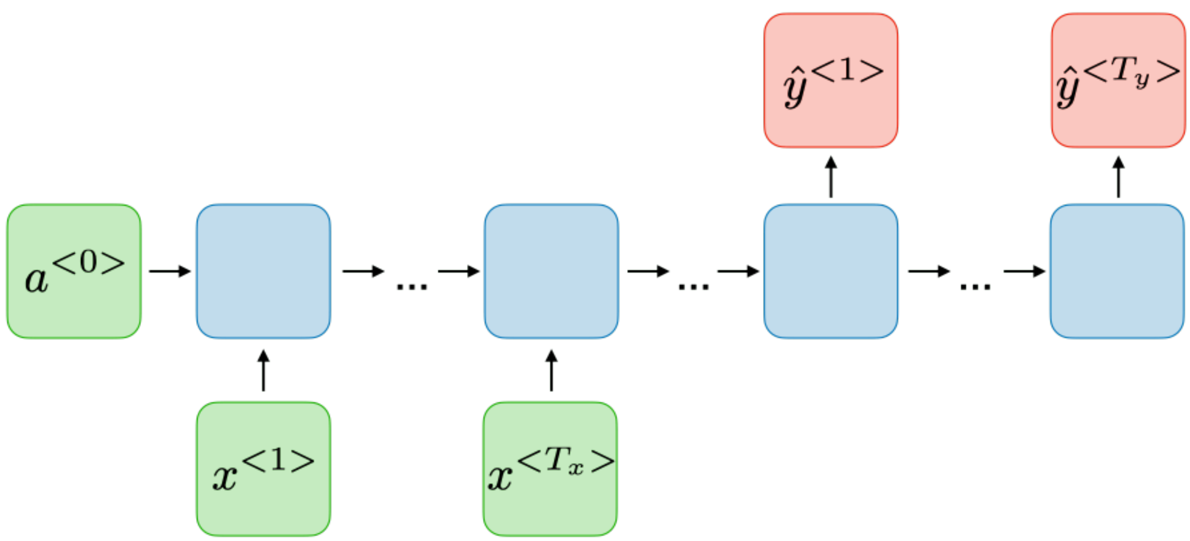


Figure 9: Many to Many, x is not equal to y

1.5 Example through pseudocode

1.5.1 initialization

$hidden\_state = [0, 0, 0, 0]$

$sentence = ["I", "like", "Deep", "Neural",]$

1.5.2 Computation

For word in sentence

$prediction, hidden\_state = rnn(word, prev\_hidden\_state)$
$prev\_hidden\_state = hidden\_state$

1.5.3 Result

$next\_word\_prediction = prediction$

1.6 Computing Hidden state

$$h_t = tanh(W_2 h_{t-1} + W_1 x_t) \tag{2}$$

Given an input vector, RNN applies tanh to update hidden state. two separate weight matrices.

Output $\hat{y}$ is a modified transformed version of this internal state.

$\hat{y}$ has separate weight matrix.

Question: Why do we use tanh in RNN? **Answer**: Because to cater the vanishing gradient problem In order to keep the gradient in the linear region of the activation function.

1.7 RNNs:Computation Graph Across Time

RNN =multiple copies of the same network. Each passes a message of h_t to it's descendants. This chaining help us in processing sequential data.

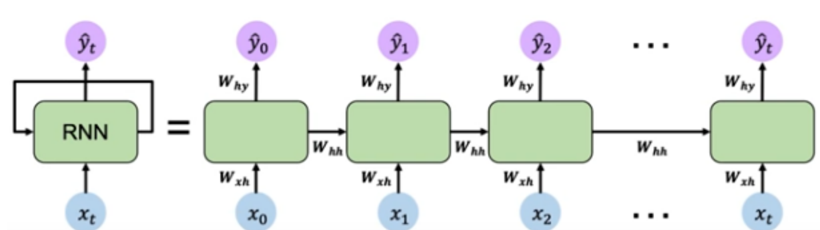


Figure 10: RNN-Chain of Layers

Forward pass will lead to output at each time step. At each time step, value for loss can be derived.

Total loss is the sum of individual loss. total loss will be used to train an RNN

1.8 Predicting the Next Word: A sequnce modling problem

Given some series for words, predict the next likely word. This morning I took my cat for a **walk**

1.9 Examples of RNN

Let us see different examples of RNN.

1.9.1 Representing language to a Neural Network

We need to convert words to numbers Used Embeddings, i.e., transform indexes into a vector of fixed size. Following process may be adopted:

1. Generate vocabulary of all possible words
2. Create indexes, ie. map words to vectors
3. One Hot Embedding. i.e.sparse vector with length equal to number of unique words in the library. ech words is encoded to a unique index in the vector.
4. Learned Embedding: i.e. take index mapping and feed our vocabulary into an index model such that similar words are mapped to the same index. The vector has lower dimension. values in the vector are learned through DNN. For instance, run and walk can be embedded to the same vector.

Recall design criteria from section 1.3.

1. handle variable length examples: the food was **great** vs. we visited a restaurant for **lunch** vs we were hungry but cleaned the house before **eating**

   FFN has input of fixed dimensionality

   RNN can handle this. how? no. of times this is processed.

2. Model long term dependency information from distant pass. RNNs can update their state through recurrence relationship
3. capture difference in sequence order. Self state of RNN depends on past history
4. Share parameters

1.10 Why parameters are shared

Parameters are shared in order to link seqquence through each time step and reduce the number of parameters For instance, let's analyse the following two sentences:

1. Yesterday, I went for a walk

2. I went for a walk yesterday

These two sentences are same. However, I went for a walk occurs at different time steps. Since parameters are shared, one only have to learn what that part means once. If parameters are not shared, one would need to learn it for every time step, where it could occur in the model.

1.11 Padding

Padding is done to have all the sequences of same length. Therefore, we will only need to construct one computational graph before starting training. We can use bucketing and padding if there is a lot of variation. For instance, 9 and 100.

This means, you pad the sequences to different bucket lengths, for example [10, 20, 50, 100]. Then, we create a computational graph for each bucket.

1.12 Back propagation Through time BPTT

1.12.1 Recall - Back Propagation Algorithm

1. Train our model in the forward direction from input to output

2. Take the derivative (gradient) of the loss with respect to each parameter

3. Shift parameters in order to minimize loss.

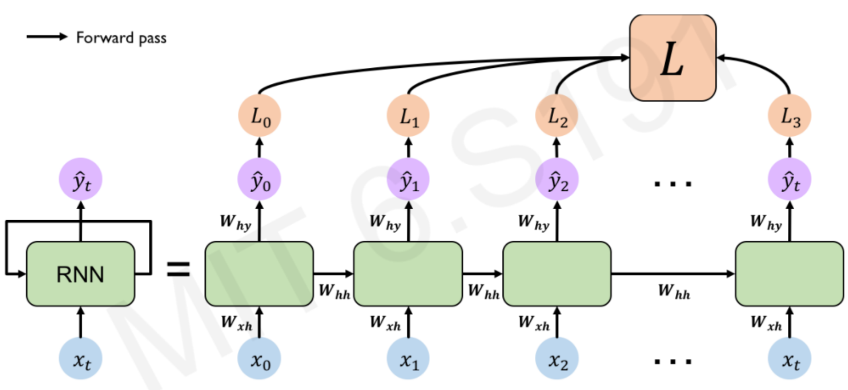


Figure 11: RNN-Loss

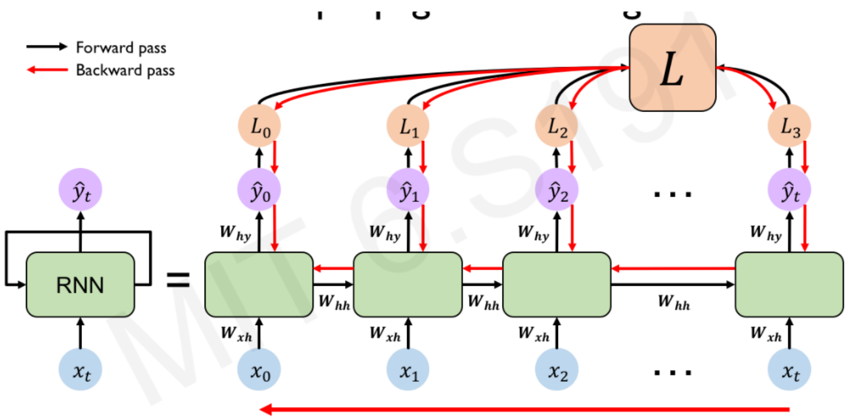


Figure 12: RNN-Loss

1.12.2 Back Propagation Through Time- RNN

1. forward through time and updating the soft state based on the input as well as previous hidden state fundamentally computing loss values at each individual time steps of the sequence.
2. Sum those individual loss to get the total loss.
3. Errors are going to be pack propagated from the overall loss through each time step from the most recent time step to the beginning of the sequence.

$$loss = \hat{y} - y \tag{4}$$

$$\frac{\partial l}{\partial \hat{y}} \tag{5}$$

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_4} \cdot \frac{\partial h_4}{\partial w_1} \tag{6}$$

$$w_1 = w_1 - \frac{\partial l}{\partial w_1} \tag{7}$$

1.13 Problems in RNN

1.13.1 Vanishing Gradient Problem

Derivative of sigmoid will always be between 0 to 1 (Why?)

During back propagation, as we move from ending layers to beginning layers, the derivative becomes very small. So the updates of weight will be in very small steps. That is, the update will be negligible. It will never be converged to the global minimum.

1.13.2 Exploding Gradient Problem

If RELU activation function is used and the derivative is greater than 1, as we move back towards initial layers the derivative will be too high. The algorithm will keep changing values and we will never achieve the global minima.

The above two problems are more persistent in long term dependencies.

1. RELU can partially solve the vanishing gradient problem . The derivative output is greater than 1 for all instances where $x > 0$.

2. Parameter initialization. Initialize weights as identity matrix to help preventing zero.

3. LSTM or GRU - control which information is passed through

2 Long Short Term Memory

A conventional RNN has two major problems. It suffers from vanishing gradient problem due to whic the learning rate becomes too slow if the depth of the network is increased. Further, it can suffer from long term dependency problem due to which longer sequences (upto 100 or 1000 sequence length) cannot be remembered. LSTM is a refined RNN, which is designed specifically to cater these two problems. LSTM networks rely on gated cell to track information throughout many time steps. Figure 13 shows the conventional RNN model, whereas figure 14 shows the LSTM version.

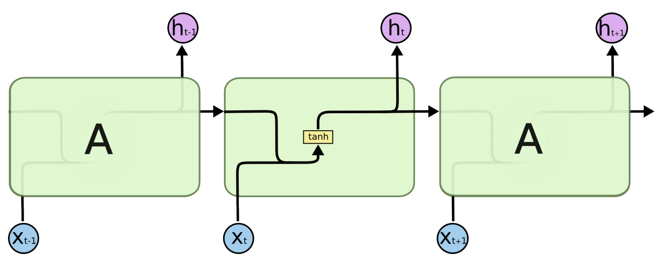


Figure 13: RNN-Model

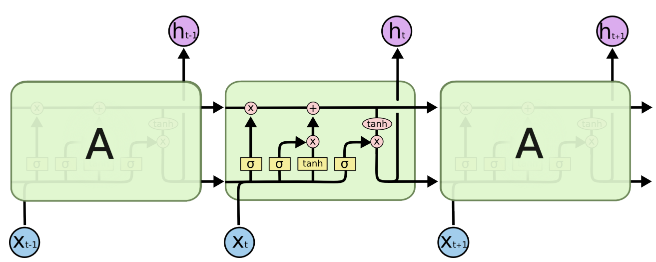


Figure 14: LSTM-Model

1. The cell state enables information to persist along the length of sequence, throughout each time step in a given LSTM cell. It is the horizontal line at the top of the cell.

2. the hidden state represents short term memory

An LSTM gate is a way to add more information to the LSTM. Figure 15 shows a sample gate. It consists of a sigmoid function and a multiplicative unit. In an LSTM, there are three of such gates. They are used to select wether information is to be added to the LSTM. The output of the sigmoid unit is between 0 to 1. This determines if the information is needed to be added. Further, there are two gates, which are based on tanh function. Together, these five gates are useful. Gates are used to regulate information flow and storage.

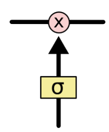
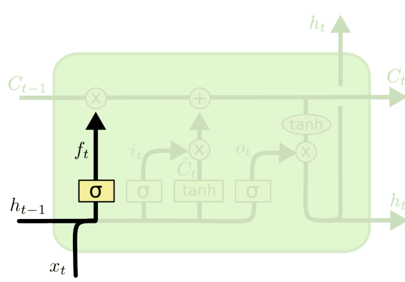


Figure 15: LSTM-Gate

Cell state is denoted by C_t.

Four stage of an LSTM network along with their figures are explained below [Ola]:

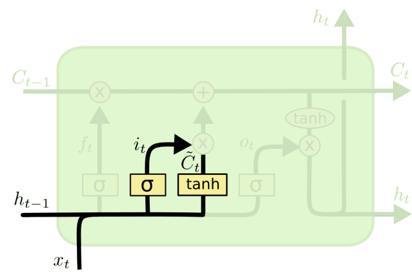
1. forget: forget irrelevant part of the previous state. Taking the previous state and passing it through sigmoid



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

Figure 16: LSTM-forget

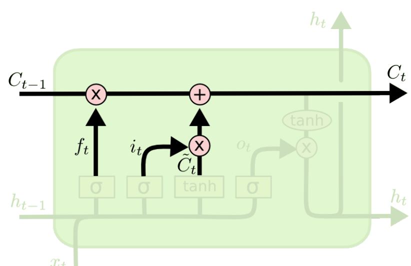
2. Store: Store most relevant new information. What part of new information and old information is relevant and store this in cell state. Take the previous state and compute unit wise multiplication of tanh and sigmoid from the previous state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 17: LSTM-input

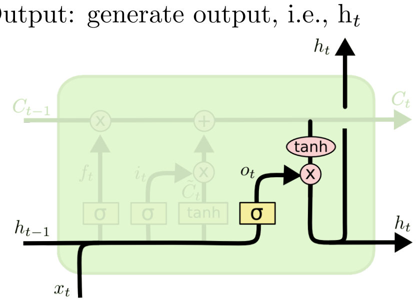
3. Update: Update internal cell state. C_t



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

LSTM-Update

Output: generate output, i.e., h_t



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM-Output
Please review this url for more information on LSTM
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3 Gated Recurrent Units

GRU or Gated Recurrent Unit is another type of sequence learning based recurrent neural network.

$$h_t = g(W_1(h_{t-1}, X_t) + b_1) \tag{8}$$

Consider a sample sentence. *The cat, which already ate ....., was full.*
The GRU unit should remember if cat is singular or plural. It will have a new memory unit called memory cell to store this information.
Let us first consider the equation, which will be used to compute the potential (or candidate value of Cell state

$\tilde{C}_t$=tanh(W$_C(C_{t-1}, X_t) + b_C$)(9)

$\Gamma_u = \sigma(W_u(C_{t-1}, X_t) + b_u)$(10)
where Γ_u denotes update gate. The sigmoid activation function that the output of the equation is either 0 or 1, 0 indicates that the information is not forwarded, while 1 indicates that the updated value is forwarded. The gate will decide whether we update $\tilde{C}_t$ GRU will memorize the value. The job of gate is to decide when to update the value.
We can now combine equations 3 and 3

$$C_t = (\Gamma_u * \tilde{C}_t) + ((1 - \Gamma_u) * C_{t-1}) \tag{11}$$

if $\Gamma_u = 1$ then the candidate value will be considered and cell state will be updated, otherwise existing value will be retained.
The cat, which already ate ....., was full.
for all the words in the middle $\Gamma_u = 0$
The candidate values and the Γ_u could be a combination of bits. In that, only some bits may be updated. e.g.
The cat, which already ate ....., was full.
some bits may be used to remember singular or plural in the word cat other bits may be used to remember ate
so only a subset of bits will be used.
We also need a relevance gate which will be denoted by Γ_r. This will determine that how relevant is (C_{t-1}) to compute the candidate value of $\tilde{C}_t$.

$\Gamma_r = \sigma(W_r(C_{t-1}, X_t) + b_r)$(12)
so equation 3 can be written as

$\tilde{\mathrm{C}}_t = tanh(W_C(\Gamma_r * C_{t-1}, X_t) + b_C)$

(13)

These equations can be written in terms of gates stages as follows:

1. **Update**

   $\Gamma_u = \sigma(W_u(C_{t-1}, X_t) + b_u)$(14)

2. **Reset**

$\Gamma_r = \sigma(W_r(C_{t-1}, X_t) + b_r)$
(15)

3. **Candidate**

$\tilde{C}_t$=tanh(W$_C(C_{t-1}, X_t) + b_C)$
(16)

4. **Cell State**

$$C_t = (\Gamma_u * \tilde{C}_t) + ((1 - \Gamma_u) * C_{t-1}) \tag{17}$$

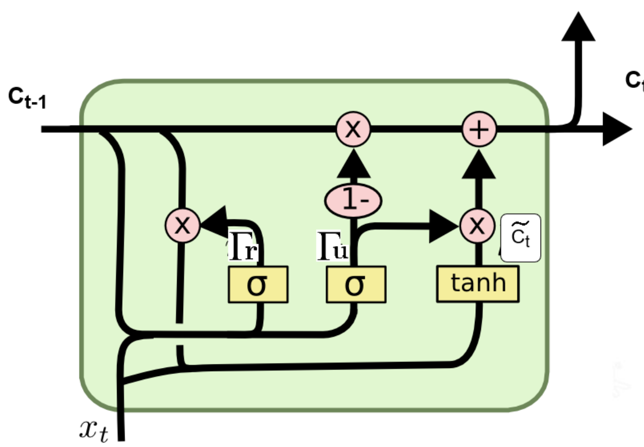


Figure 18: Gated Recurrent Unit

Figure 18 shows the overall diagram of GRU.

4 Bi Directional RNN

These are sequence learning networks which can traverse in both the directions, i.e. forward and backward. Consider the following sentences

He said, "Teddy bears are on Sale!" He said, "Teddy Roosevolt was a great President!"

For both these sentences, the first three words are same. However, the fourth word can be predicted correctly, if backward traversal is used.

At time t=3, Teddy could be a person or an animal. Therefore, we need to look ahead.

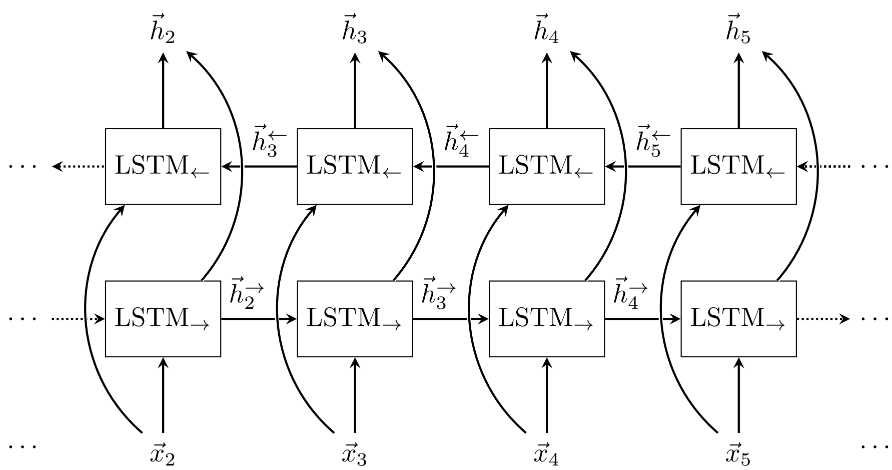
A bi-directional Sequential network could be RNN/LSTM/GRU

It has a forward path and a backward path

It forms an acyclic graph - a forward network and a backward network. The neural network will utilize both the forward and the backward activations.

for instance, for prediction at time t=3, forward propagation will be used for time t_1 i.e., X_1 and t_2 i.e. X_2, whereas backward propagation will be used from time t_4, i.e. X_4

$\hat{y}_t$=g(W$_y$($\overrightarrow{h_t}, \overleftarrow{h_t}$) + b_y)(18)



5 RNN Example - Machine Translation

In this section, we will use Encoder/Decoder Architecture to learn Machine Translation.

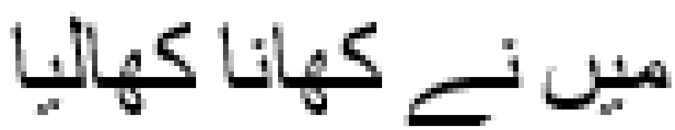
5.1 Encoder - Decoder

Encoder Decoder Architecture allows encoding of input into a latent space and using this latent information to decode into a new transformed output. Latent space here could be considered as an abstract lower-dimensional representation of high-dimensional data. Figure 5.1 illustrates the architecture.

Figure 19: Encoder Decoder Architecture

5.2 Encoder - Decoder - Machine Translation

To understand machine translation, let's consider an example where text in urdu is needed to be translated to English. Figure 5.2 shows the input text to be translated to English



scalescale

Figure 20: Urdu Input

Figure 5.2 shows the overall architecture. An important point to note that the major difference between training and testing phases is teacher forcing. That is, during the training phase, the decoder get's the input as the actual output (and not the predicted output) at the previous time step. So even if the predicted output was in correct, the actual output would be sent as an input. This is how we can train the network. In comparison, even in the testing phase, the predicted output from the previous time step is sent as an input.

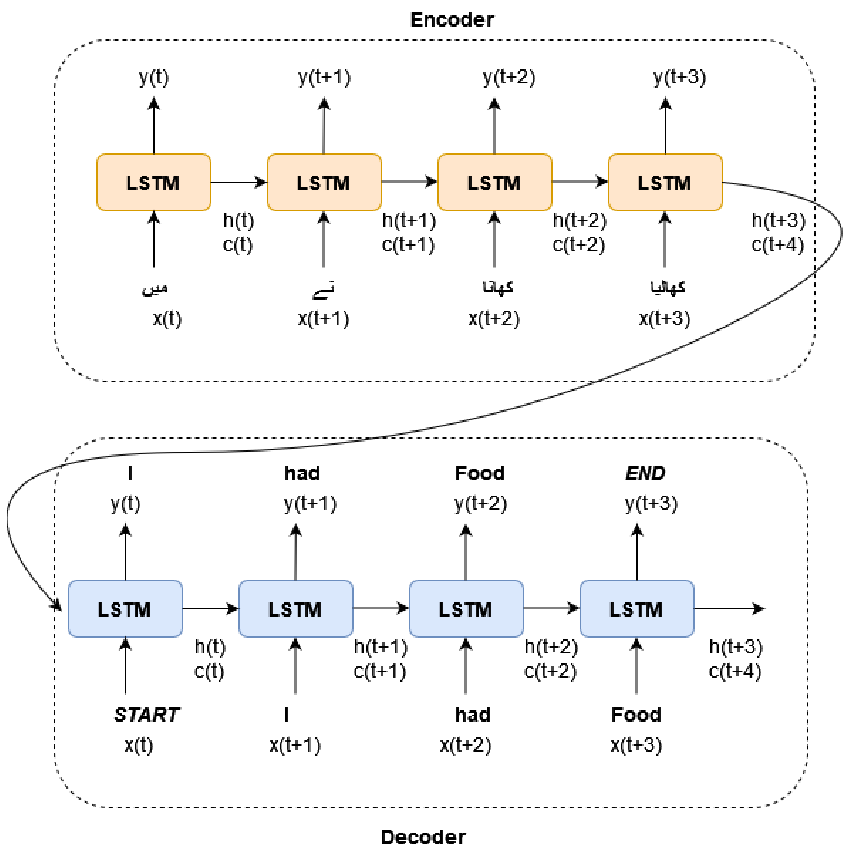


Figure 21: Encoder Decoder -Machine Translation

References

[OCW] MIT OCW. *MIT opencourseware MIT 6.s191. Introduction to Deep Learning.* URL: `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-s191-introduction-to-deep-learning-january-iap-2020/`. (accessed: 01.01.2022).

[Ola] Christopher Olah. *Understanding lstm networks.* URL: `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).