**CS 217: Artificial Intelligence and Machine Learning** **Jan-Apr 2024**

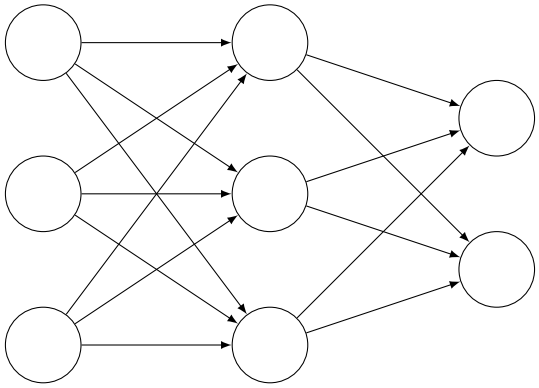
# Lecture 13: Recurrent Neural Networks

*Lecturer: Swaprava Nath* *Scribe(s): SG25, SG26*



## 13.1 Introduction to Recurrent Neural Networks (RNNs)

In our earlier lectures, we delved into the workings of regular feedforward Neural Networks. These networks take a $d$-dimensional input and calculate an output based on specified dimensions. They update weights and biases between layers without considering the order of input training data. This framework is particularly useful for classification tasks, such as predicting whether a student will pass or fail a test by taking attendance, background and several other factors as input.



However, applications like predicting the next word in autofill sentences or identifying grammatical errors and spelling corrections demand consideration of the order of input elements. In such scenarios, Recurrent Neural Networks (RNNs) play a vital role.

Sequential data, as encountered in these applications, is characterized by two key aspects: the order of inputs matters, and the length of input is variable. RNNs are specifically designed to handle such sequential data.

## 13.2 Basic RNN structure

RNNs are designed to handle **sequences**. This can be used to analyze video (sequences of images), writing/speech (sequences of words), etc. LSTMs are designed to let important information persist over time. RNNs will often "forget" over time. **FFNNs** are **memoryless systems**; after processing some input, they forget everything about that input. The essential difference between the two is the **self loop** within the hidden layer which is useful in capturing the unseen temporal relationship that might exist in the training data.

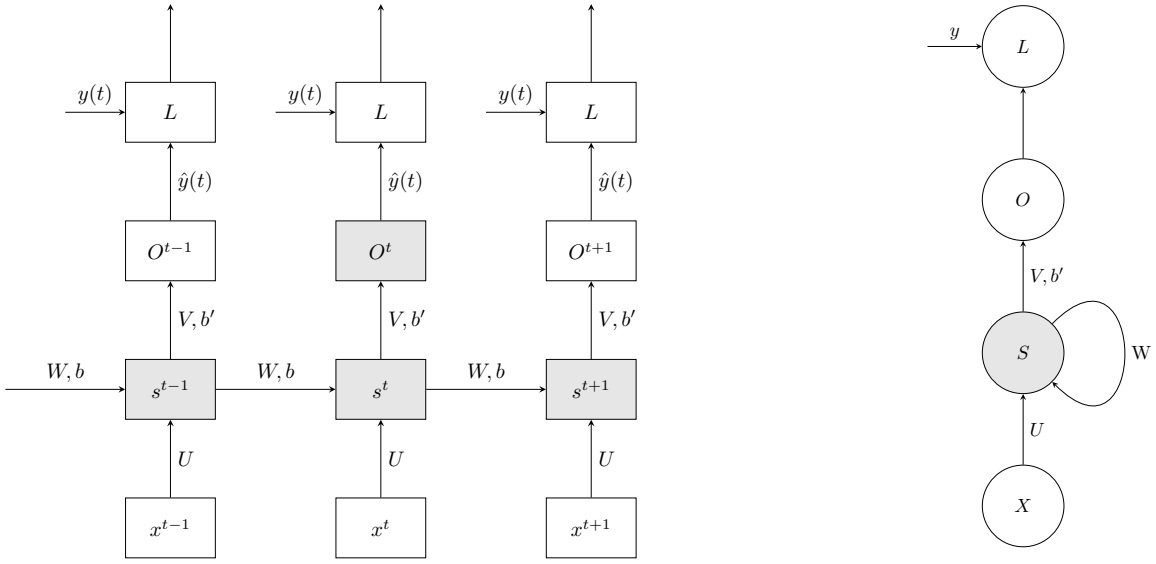



Figure 13.1: Architecture of an RNN

$$
\begin{aligned}
s^{(t)} &= g(Ws^{(t-1)} + Ux^{(t)} + b) \\
o^{(t)} &= Vs^{(t)} + b' \\
\hat{y}^{(t)} &= \text{softmax}(o^{(t)})
\end{aligned}
$$

Where g can be any activation function like sigmoid, `tanh`, ReLU(Rectilinear Linear Unit).

**Named Entity Recognition:** Named Entity Recognition (NER) using Recurrent Neural Networks (RNNs) is a popular approach for extracting entities such as names of people, organizations, locations, dates, etc., from text data. ***Example***: Somesh worked at Cisco in Banglore.

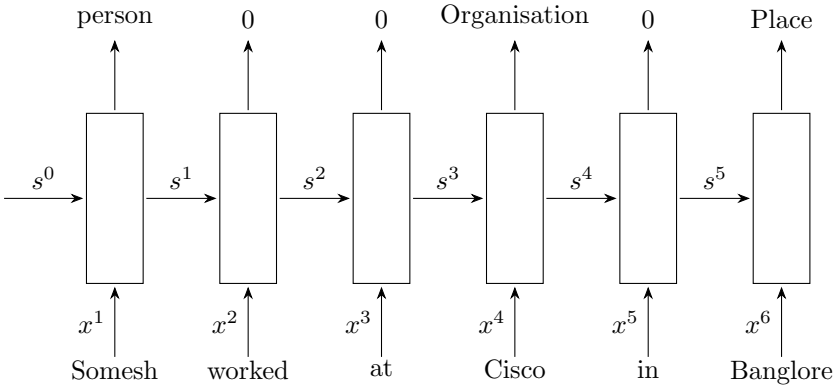
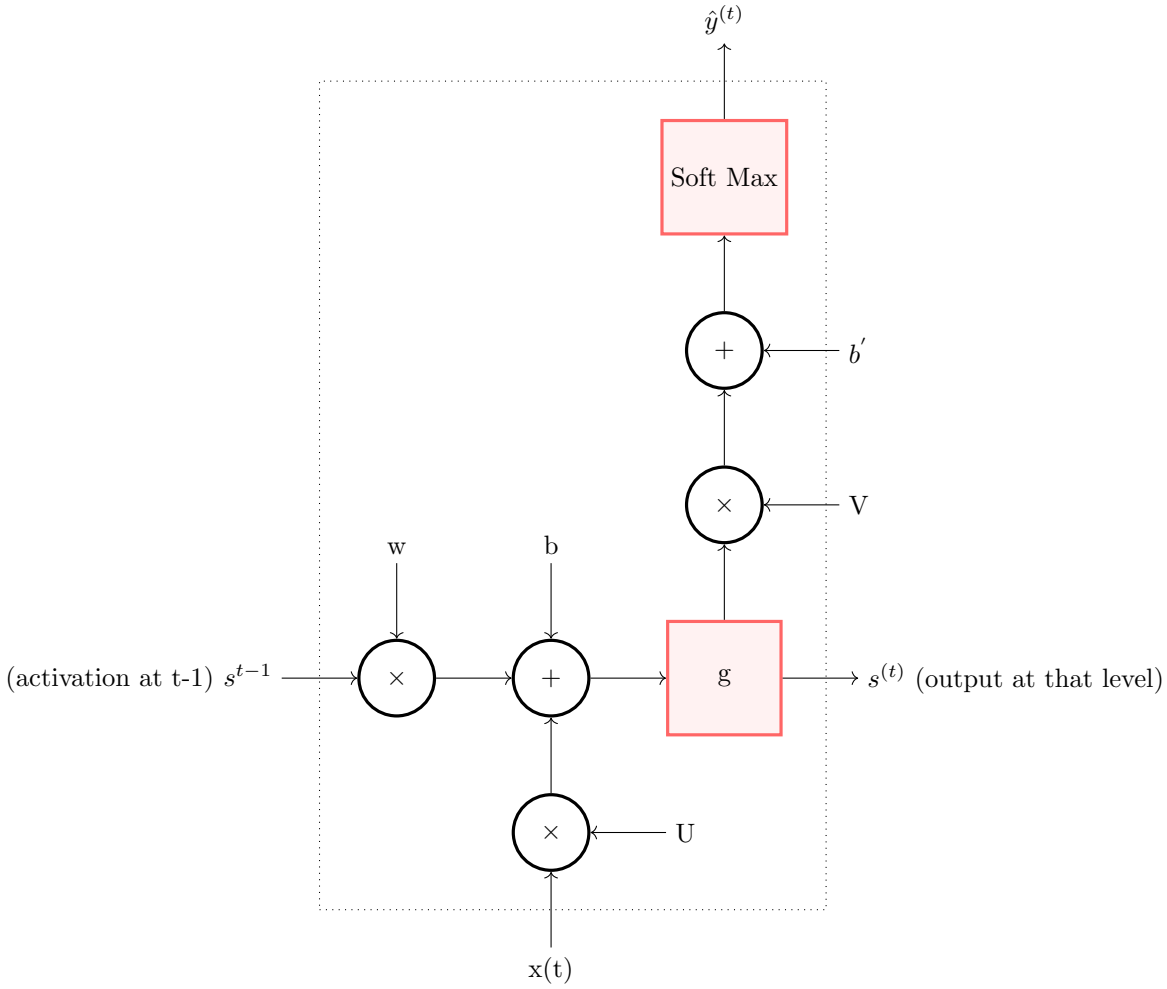



Figure 13.2: Named Entity Recognition

## 13.3 RNN Possible structures

Each cell of RNN can be represented as:

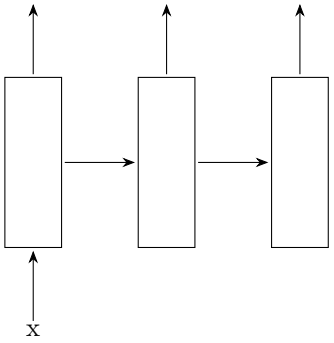



In the realm of word representation, we encounter two primary methodologies:

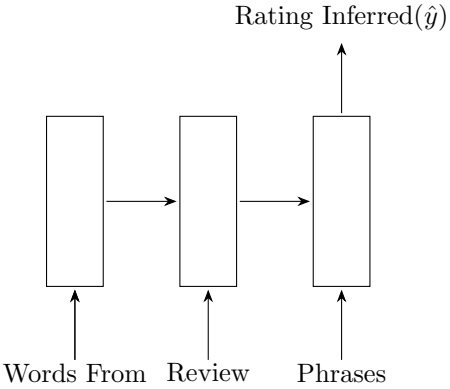
1. **One-Hot Encoding:** This approach represents each word as a sparse vector, where the length of the vector equals the total vocabulary size. Each word is denoted by a vector with all elements set to 0, except for the element corresponding to the word, which is set to 1. Despite its simplicity, this method is deemed inefficient due to the high dimensionality of the resulting vectors.

2. **Word Embeddings:**In contrast, word embeddings offer a more efficient representation. Here, words are encoded as dense vectors of lower dimensionality compared to one-hot encoding. These vectors, trained using techniques like Word2Vec or GloVe, capture semantic relationships between words based on their contextual usage, which will be studied in later courses. This dense representation not only conserves memory but also enhances the model's ability to capture nuanced semantic meanings.

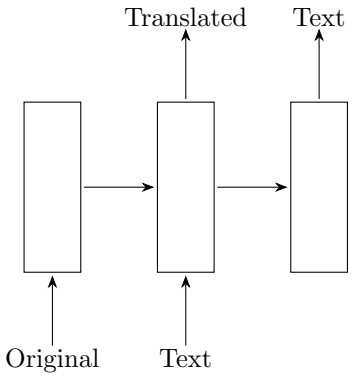

(a) FFNN(RNN without dependencies)



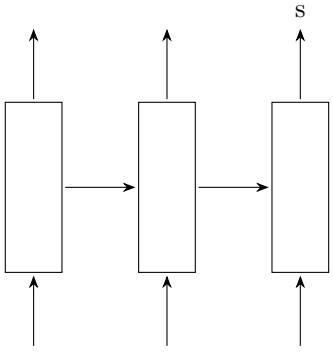

(b) One-Many(multiple outputs,single input). A common application is Image Captioning.




(c) Many-One. This type is characterized by having single output and multiple inputs. A common application is predicting ratings by taking reviews written by users as input.




(d) Many-Many Staggered(Length of input(x) and output($\hat{y}$) not the same). In this type of RNN, both input and output are many but the length of input and output need not be equal. A common example being text translation into some other language where the output generated starts at a farther cell rather than the input started due to some processing time.




(e) Many-Many Synchronous(prediction for every step of the sequence), used in Named-Entity Recognition(NER),each cell taking one input and giving one output by considering the effect of previous inputs.

Figure 13.3: Various architectures

There are multiple possible structures for a Recurring Neural Network. Some of which are described in the above figure.

## 13.4 Training RNN's

During RNN training, humans label each word with its corresponding entity type, such as person, place, organisation...etc then use these as training data. The purpose of training an RNN is to determine the weights that we need to use within the cell. If we were to use different weights for each cell, as we do in a normal feedforward neural network with different weights at each layer, we would face two main problems that we must overcome when training the RNN:

- **Variable length of the dataset:** Let's consider a scenario where the longest number of parameters for input variable x during training is 15. This means that we have a framework with 15 cells while training. If the neural network encounters a parametric x with 16 parameters, we need to retrain it to deal with 16 cells. This involves creating new weights to connect the 15 existing cells to the 16th cell. Unfortunately, this is a significant issue as we can never predict the size of x. To overcome this challenge, we prefer to use the same weights across all cells.
- **Overfitting of the model:** Say we are using n rounds in our framework. Using different weights in cells in different rounds will just increase the complexity of the neural network. This results in the overfitting of training data.

So we use the same weight vectors at each round as it is optimal for both overfitting and length problems.

### 13.4.1 Forward Pass

The standard functions we use are:

$$\mathbf{g} = Softmax$$

$$\mathbf{S} = \tanh$$

using the categorical cross-entropy function as a loss function

$$Loss(y, \hat{y}) = -\sum y_k \log \hat{y}_k$$

$$softmax = -\sum_{k=1}^{K} I\{y = k\} \log(\hat{y}_k)$$

where $\hat{y}_k$ is of the form $\frac{e^{w_1 x}}{\sum_i e^{w_i x}}$

Forward Pass is computing $s^{(1)}, s^{(2)}, ....s^{(t)}$ and $\hat{y}$

$$s^{(t)} = \tanh\left(Ws^{(t-1)} + Ux^{(t)} + b\right)$$

$$\hat{y} = softmax\left(Vs^{(t)} + b^{'}\right)$$

### 13.4.2 Backpropagation Through Time (BPTT)

To find: $\frac{\partial L}{\partial w}$ , $\frac{\partial L}{\partial b}$ , $\frac{\partial L}{\partial U}$ , $\frac{\partial L}{\partial V}$ , $\frac{\partial L}{\partial b'}$
We know that

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left( - \sum y_k \log \hat{y}_k \right)$$

### 1.Computing $\frac{\partial L}{\partial V}$

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial V}$$

$\frac{\partial \hat{y}}{\partial V}$ can be computed since $\hat{y} = softmax(VS^{(t)} + b^{'})$

### 2.Computing $\frac{\partial L}{\partial b'}$

$$\frac{\partial L}{\partial b'} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial b'}$$

$\frac{\partial \hat{y}}{\partial b'}$ can computed since $\hat{y} = softmax(VS^{(t)} + b^{'})$

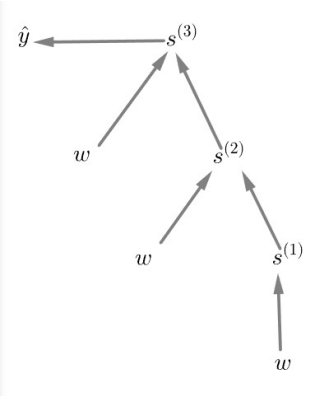
### 3.Computing $\frac{\partial L}{\partial w}$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

If f is a function depending of $g_1$(x) and $g_2$(x) then

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g_1} * \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} * \frac{\partial g_2}{\partial x}$$

Hence for the case with three cells in RNN:

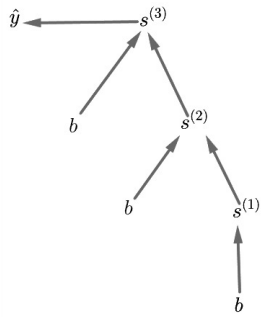



$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} * \left( \frac{\partial \hat{y}}{\partial s^{(3)}} \right) * \frac{\partial s^{(3)}}{\partial w} + \frac{\partial L}{\partial \hat{y}} * \left( \frac{\partial \hat{y}}{\partial s^{(3)}} * \frac{\partial s^{(3)}}{\partial s^{(2)}} \right) * \frac{\partial s^{(2)}}{\partial w} + \frac{\partial L}{\partial \hat{y}} * \left( \frac{\partial \hat{y}}{\partial s^{(3)}} * \frac{\partial s^{(3)}}{\partial s^{(2)}} * \frac{\partial s^{(2)}}{\partial s^{(1)}} \right) * \frac{\partial s^{(1)}}{\partial w}$$

In the above summation, terms in the brackets are of the form $\frac{\partial \hat{y}}{\partial s^{(i)}}$
So for the case with T cells in RNN:

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{T} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s^{(i)}} * \frac{\partial s^{(i)}}{\partial w}$$

## 4.Computing $\frac{\partial L}{\partial b}$




$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial b}$$

For the case with three cells in RNN(from the above figure):

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} * \left(\frac{\partial \hat{y}}{\partial s^{(3)}}\right) * \frac{\partial s^{(3)}}{\partial b} + \frac{\partial L}{\partial \hat{y}} * \left(\frac{\partial \hat{y}}{\partial s^{(3)}} * \frac{\partial s^{(3)}}{\partial s^{(2)}}\right) * \frac{\partial s^{(2)}}{\partial b} + \frac{\partial L}{\partial \hat{y}} * \left(\frac{\partial \hat{y}}{\partial s^{(3)}} * \frac{\partial s^{(3)}}{\partial s^{(2)}} * \frac{\partial s^{(2)}}{\partial s^{(1)}}\right) * \frac{\partial s^{(1)}}{\partial b}$$

So for the case with T cells in RNN:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{T} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s^{(i)}} * \frac{\partial s^{(i)}}{\partial b}$$

## 5.Computing $\frac{\partial L}{\partial U}$

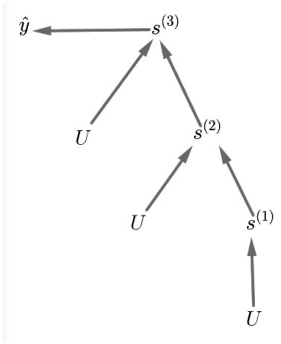
$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial U}$$

For the case with three cells in RNN:

$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial \hat{y}} * \left(\frac{\partial \hat{y}}{\partial s^{(3)}}\right) * \frac{\partial s^{(3)}}{\partial U} + \frac{\partial L}{\partial \hat{y}} * \left(\frac{\partial \hat{y}}{\partial s^{(3)}} * \frac{\partial s^{(3)}}{\partial s^{(2)}}\right) * \frac{\partial s^{(2)}}{\partial U} + \frac{\partial L}{\partial \hat{y}} * \left(\frac{\partial \hat{y}}{\partial s^{(3)}} * \frac{\partial s^{(3)}}{\partial s^{(2)}} * \frac{\partial s^{(2)}}{\partial s^{(1)}}\right) * \frac{\partial s^{(1)}}{\partial U}$$

So for the case with T cells in RNN:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{T} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s^{(i)}} * \frac{\partial s^{(i)}}{\partial U}$$




## 13.5 Vanishing Gradient Problem

In simpler terms, the vanishing gradient problem happens when the gradients become smaller and smaller as they move through the layers of a neural network during training. This makes it difficult to update the weights in the network, which can cause the learning process to be slow or even stop altogether.

One of the reasons why this happens is because of the use of certain functions, like `tanh` and sigmoid, which tend to saturate at certain values.

In the case of RNN, the problem is further complicated because the partial derivative of weights for each time step depends on the effect of all the previous time steps. As the sequence of time steps increases, the gradient terms from earlier time steps decrease exponentially as they are multiplied by gradient terms whose values are less than one at each later time step up to the $t^{th}$ time step, which means that their influence on $t^{th}$ time step diminishes rapidly as t increases. This results in the decrease of the influence of earlier time steps as the sequence length increases. This can make it difficult for the network to learn from longer sequences of data, which can be a problem in certain applications.

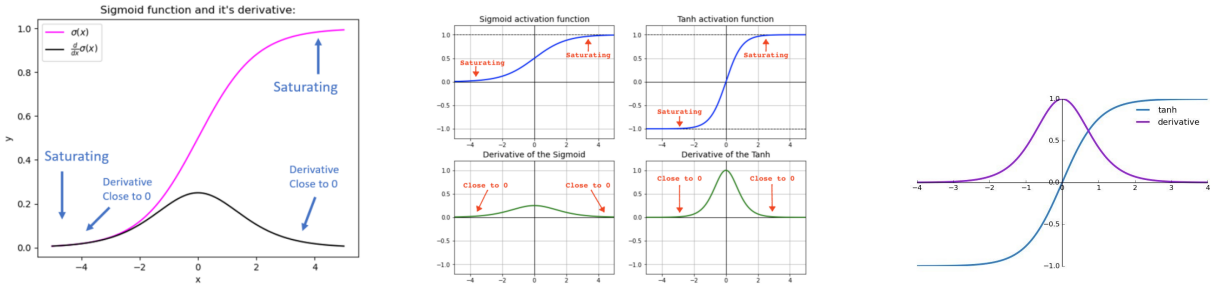
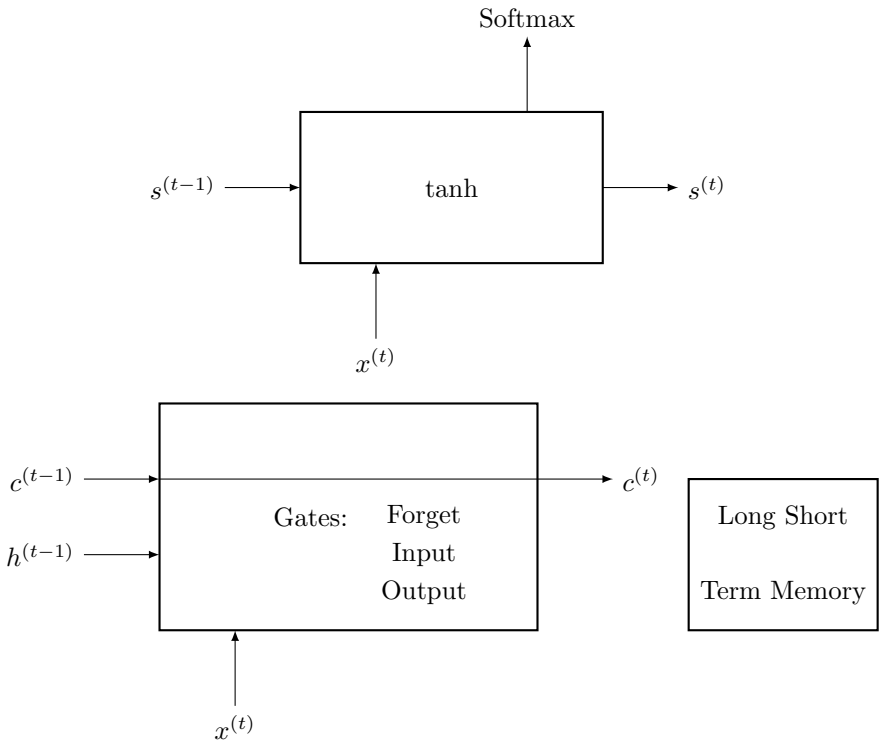



Figure 13.4: **Saturating functions:**tanh and sigmoid

Consider the statement,

*Isha* went for a walk, . . . *He* was overjoyed

In traditional RNNs, the grammar checker won't be able to detect this change from *Isha* to *He* and the error will remain. To mitigate this a modification to the cell structure is Long Short Term Memory Networks, we modify the cell structure in the following way:




Where $c^{(i)}$ are memory cell states and $h^{(i)}$ are hidden cell states.