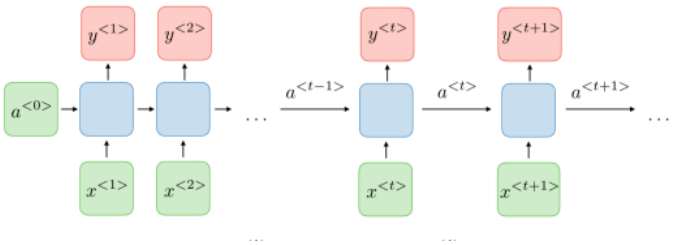
Recurrent Neural Networks

R Basheer Ahammad

1 Architecture

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:

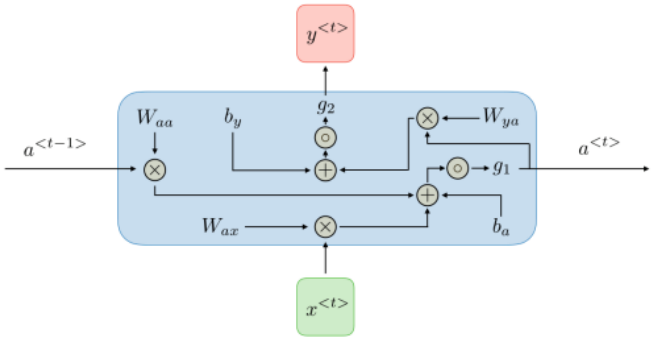


For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally, and g_1 and g_2 are activation functions.



The pros and cons of an RNN are shown in below table:

Advantages	Drawbacks
- Possibility of processing input of any length - Model size not increasing with size of input - Computation takes into account historical information - Weights are shared across time	- Computation being slow - Difficulty of accessing information from a long time ago - Cannot consider any future input for the current state

Loss Function

In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time steps is defined based on the loss at every time step as follows:

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

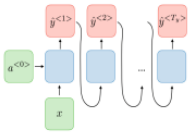
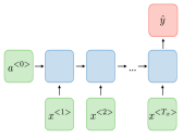
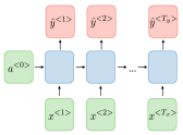
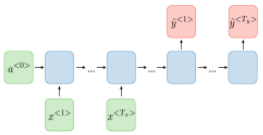
Backpropagation through time

Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix $\mathbf{W}$ is expressed as follows:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}} = \sum_{t=1}^{T} \left.\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}}\right|_{(t)}$$

Applications

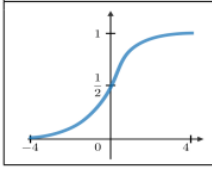
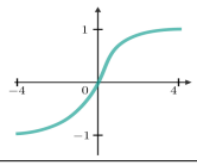
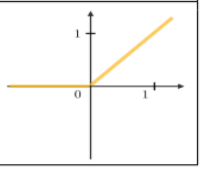
RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

2 Handling Long term dependencies

Commonly used activation functions

The most common activation functions used in RNN modules are described below:

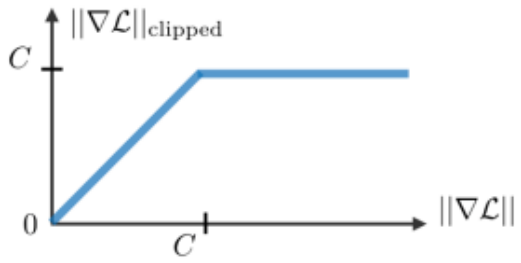
Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$
		

Vanishing/Exploding Gradient

The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

Gradient Clipping

It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



Types of gates

In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted as Γ and are equal to:

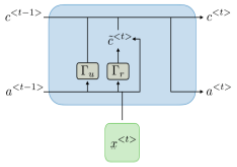
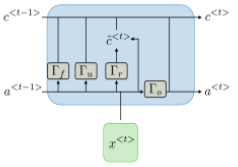
$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

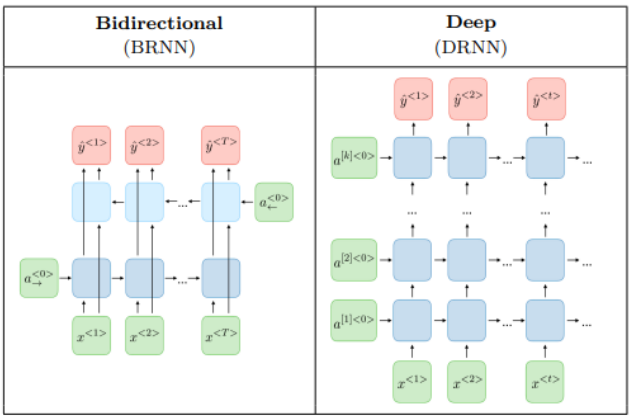
where W, U, and b are coefficients specific to the gate, and σ is the sigmoid function. The main ones are summarized in the table below:

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

GRU/LSTM

– Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		



Variants of RNN

The table below sums up the other commonly used RNN architectures: