Architecture of RNNs

Mohammed Luqmaan Akhtar

January 29, 2026

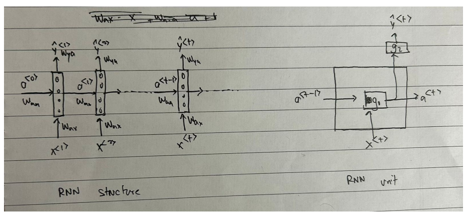
Sequence Data and Models

Sequential data is a type of data where the order of observations matters. Each data point is part of a sequence, and the sequence's integrity is crucial for analysis. There are various applications of analysis of sequential data:

- Speech recognition: sound wave to words
- Music: any symbol to music wave
- Sentiment classification: "Movie is bad" to 2
- DNA sequence analysis
- Machine translation: French to English
- Video activity recognition
- Named entity recognition

We cannot use conventional Neural Networks for this because both the input and outputs can be of different lengths and because it does not share features across different positions of the text.

Recurrent Neural Networks



Let us consider a single training example. Suppose the task is of named entity recognition, so for each input sequence there is an output either 0 or 1. There are three main components the hidden state, the input sequence's current iteration and the output.

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

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

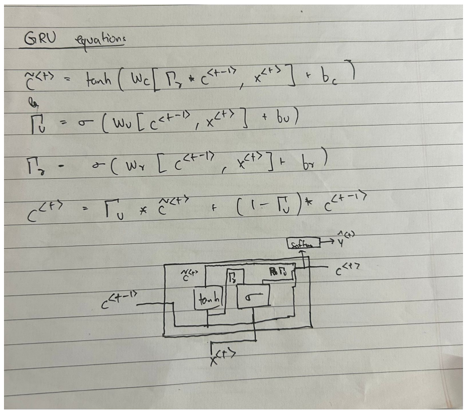
$a^{<t>}$ is the hidden state which stores the context long term, $x^{<t>}$ is the current iteration of the sequence and $y_p^{<t>}$ is the predicted output. The weights and biases are like that of a conventional neural network. A single RNN cell can be treated as a single neural network layer and the RNN keeps inputting sequence values into the same layer along with the previous hidden state and getting the output values. The computation of weights and biases is done by using optimization algorithms with the gradients calculated with respect to a chosen loss function. The loss function is chosen according to the task with Binary Cross Entropy Loss for binary classification, Negative Loss Likelihood for multiple classification problems etc.
There are different types of RNN architectures:

- Many to Many: named entity recognition
- Many to One: sentiment classification
- One to One: This is just like a conventional Neural Network though.
- One to Many: music generation

Vanishing Gradient Problem

The problem with RNNs is that across a very long sequence the hidden state tends to explode or vanish. RNNs need to hold information across very long term dependencies so this is a big problem. The exploding gradient problem can be easily fixed by gradient clipping the vanishing gradient problem is hard to solve. The concepts of GRU and LSTM help solve this very problem.

Gated Recurrent Unit



GRU equations

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

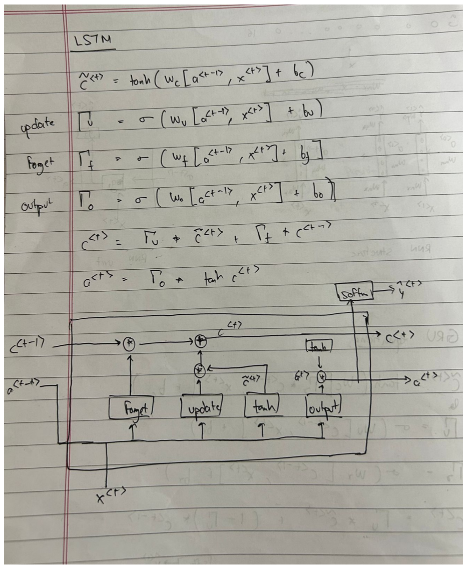
$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

softmax
$\hat{y}^{<t>}$
$\tilde{c}^{<t>}$
Γ_r
Γ_u
$c^{<t>}$
$c^{<t-1>}$
tanh
σ
$x^{<t>}$

c is the memory cell. It can hold information across long sequences. we calculate a $c^{`}$ which is a potential candidate to replace the current value of the memory cell. And according to the update gate value it according updates the value of memory cell. It has another gate which is used to compute the relevancy of the previous memory cell in competing the candidate for replacement. GRU has significantly less vanishing gradient problem because even across many iterations the value of c can remain practically changed if the update gate's value is clost to 0.

Long Short Term Memory



LSTM

$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$

update $\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$

foget $\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f]$

output $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o]$

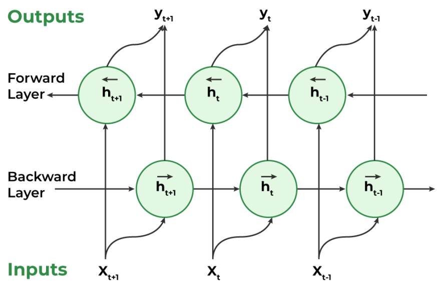
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$

$a^{<t>} = \Gamma_o * \tanh c^{<t>}$

Long Short Term Memory is another solution to the vanishing gradient problem. It is more complex and has more gates than GRU. The output gates decides how much of the memory is exposed in the current iteration. Also, if the forget and update values are adjusted well it can remember the memory for a long time.

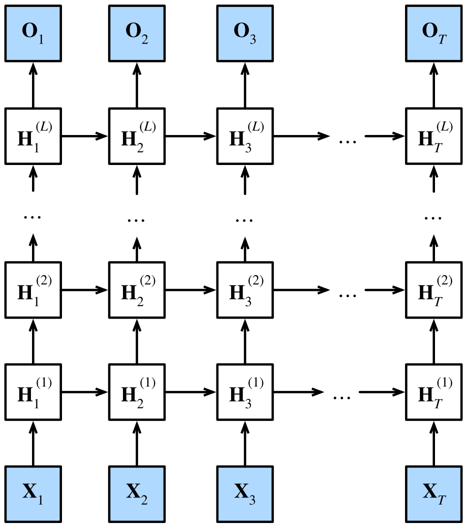
Bidirectional RNN

In named entity recognition if the two sentences are:
He said, "Teddy bears are on sale!"
He said, "Teddy Roosevelt was a great President!"
We need context from both sides of the word to classify whether Teddy is a name or an item.



Deep RNN

RNNs stacked over each other and connected are called Deep RNNs. The final classification layer can be independent Fully Connected layers before the output.


$\mathbf{O}_1$
$\mathbf{O}_2$
$\mathbf{O}_3$
$\mathbf{O}_T$
$\mathbf{H}_1^{(L)}$
$\mathbf{H}_2^{(L)}$
$\mathbf{H}_3^{(L)}$
$\mathbf{H}_T^{(L)}$
$\mathbf{H}_1^{(2)}$
$\mathbf{H}_2^{(2)}$
$\mathbf{H}_3^{(2)}$
$\mathbf{H}_T^{(2)}$
$\mathbf{H}_1^{(1)}$
$\mathbf{H}_2^{(1)}$
$\mathbf{H}_3^{(1)}$
$\mathbf{H}_T^{(1)}$
$\mathbf{X}_1$
$\mathbf{X}_2$
$\mathbf{X}_3$
$\mathbf{X}_T$