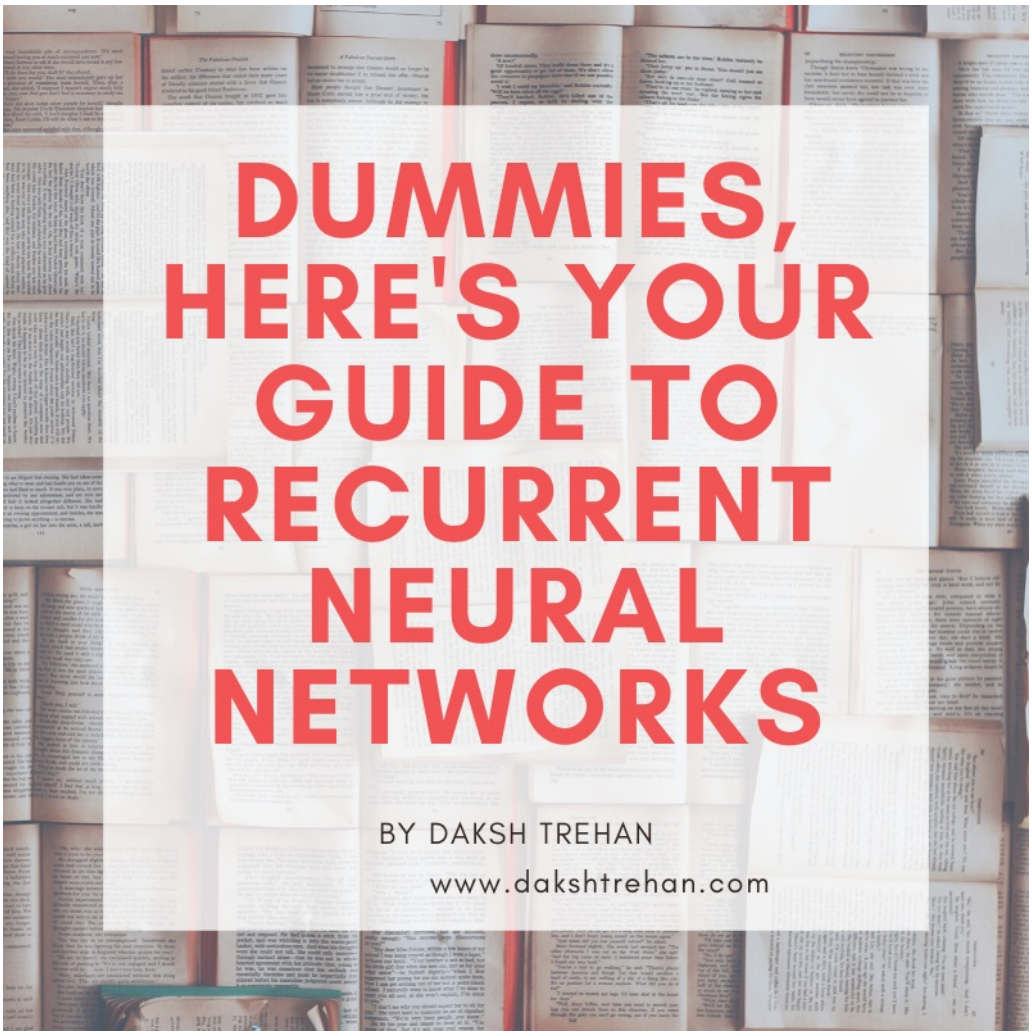
Deep Learning, Natural Language Processing

Recurrent Neural Networks for Dummies

A perfect guide to Recurrent Neural Networks



You asked Siri about the weather today, and it brilliantly resolved your queries.

But, how did it happen? How it converted your speech to the text and fed it to the search engine?

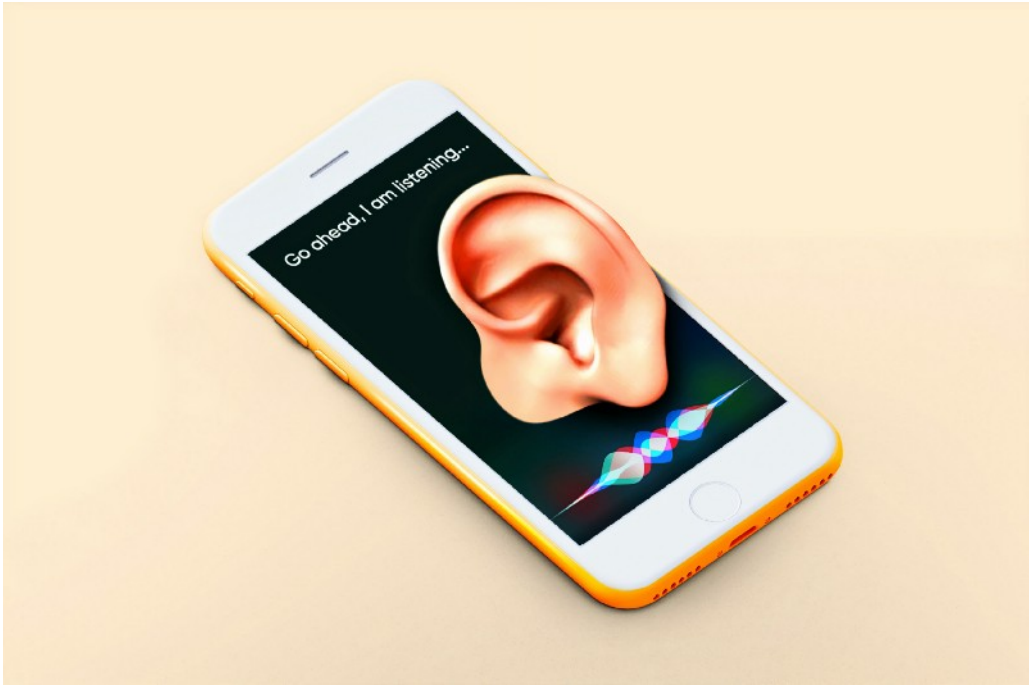


Photo by Morning Brew on Unsplash

This is the magic of **Recurrent Neural Networks.**

Recurrent Neural Networks(RNN) lies under the umbrella of **Deep Learning.** They are utilized in operations involving Natural Language Processing. Nowadays since the range of AI is expanding enormously, we can easily locate Recurrent operations going around us. These play an important role ranging from *Speech Translation, Music Composition to predicting the next word in your mobile's keyboard.*

The types of problems that RNN caters to are:

- Outputs are dependent on previous inputs. (Sequential Data)
- The length of the input isn't fixed.

Sequential Data

Photo by Erik Mclean on Unsplash

To understand Sequential data, let us suppose you have a dog standing still.

Now, you're supposed to predict in which direction will he move? So with only this limited information imparted to you, how would you do this? Well, you can irrefutably take a guess, but in my opinion, what you'd come up would be a random guess. Without knowledge of where the dog has been, you wouldn't have enough data to predict where he'll be going.

Photo by Marcus Benedix on Unsplash

But, now if the dog starts running in a particular direction and if you try to record the movements of dogs, you'll be pretty sure the directions he'll be choosing. Because at this instant you've enough information to make a better prediction.

So a sequence is a particular order in which one thing follows another. With this information, you can now see that the dog is moving towards you.

Text, Audio are also illustrations of sequence data.

When you're talking to someone, there is a sequence of the words you utter. Similarly, when you e-mail someone, based on your texts, there is some certainty about what your next words would be.

Sequential Memory

As mentioned earlier, RNNs cater to the problems that involve inter-dependency between outputs and previous inputs. That indirectly means, there is some memory affiliated to these kinds of Neural Networks.

Sequential memory is something that helps RNN achieve its goal.

As to better understand, I would ask you to recall the alphabet in your head.

Photo by Jessicah Hast on Unsplash

That was an easy task, if you were taught this specific sequence, it should come quickly to you.

Now, if I ask you to recall alphabets in a reverse manner.

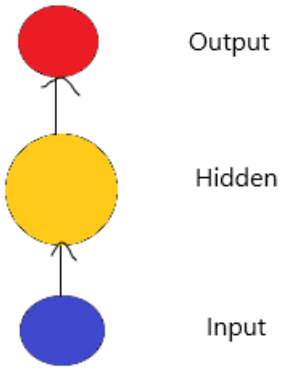
Z Y X W V U T S R Q P O N
M L K J I H G F E D C B A

I bet this task is much solid. And in my opinion, it will give you a hard time.

So, the reason the former task proved to be resilient because you've learned the alphabets as a sequence. Sequential memory makes it easier for your brain to recognize patterns.

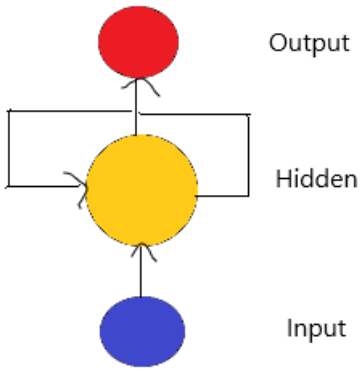
How Recurrent Neural Networks differ from Neural Networks?

As discussed earlier, Recurrent Neural Network comes under Deep Learning but so does Neural Networks. But due to the absence of an internal state, Artificial Neural Networks are not something that we use to process our sequential data.



Feedforward Network

To develop a Neural Network that is robust for Sequential data, we add an internal state to our feedforward neural network that provides us with internal memory. Or in nutshell, Recurrent Neural Network is a generalization of a feedforward neural network that has internal memory. **RNN implements the abstract concept of sequential memory**, that helps them by providing the previous experience and thus allowing it to predict better on sequential data.



Recurrent Neural Network

RNN proves it recurrent nature by performing the same function for every input, while the output of current input depends upon the past input. Comparing it to Feedforward Neural Network, in RNN, all the inputs are inter-dependent on each other unlike that in vanilla form.

Working of RNN

Okay, but how does RNN replicate those internal memories and actually work?

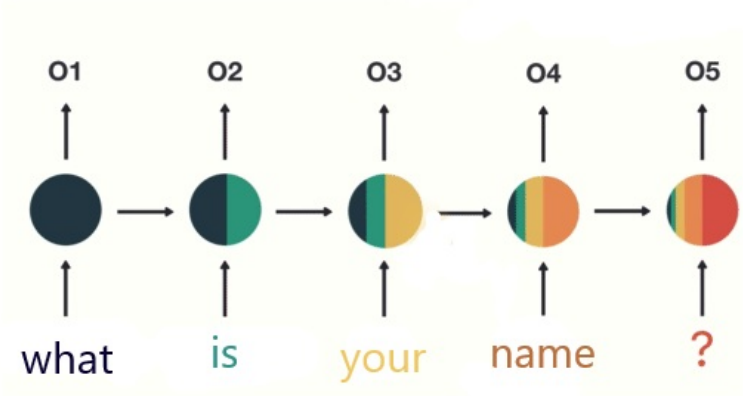
Suppose, a user asked, "*What is your name?*"

Since RNN solely depends upon sequential memory, we expect our model to break up the sentence into individual words.

At first, “What” is fed into RNN. Our model then encodes it and presents us with an output.

For the next part, we feed the word “is” and the former output that we got from the word “What”. RNN has now access to the information imparted by both words: “What” and “is”.

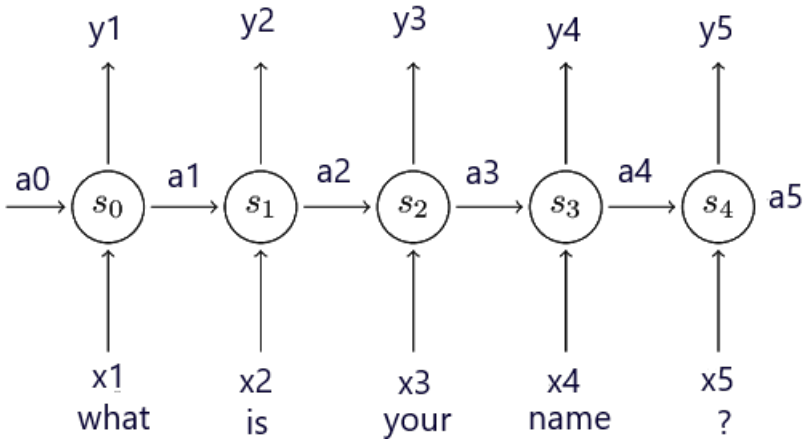
The same process will be iterated until we reach the end of our sequence. And In the end, we can expect RNN had encoded information from all the words present in our sequence.



What is your name?

Since the last output is developed by combining the former outputs and the last input, we can pass the final output to the feedforward layer to achieve our goal.

To create the context, let us resemble **input by x**; **output by y**; and **state vector by a.**



When we pass our first input i.e. x0 (“What”), we are provided with the output y1 and a state vector a1, that is passed to next function s1 to accommodate the past output of x0.

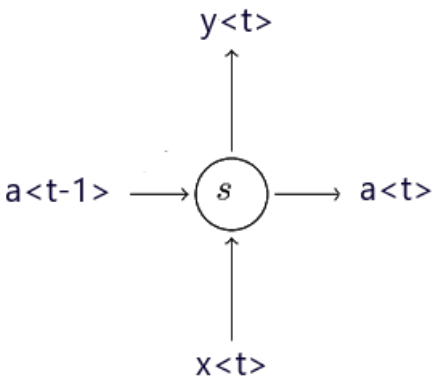
The process iterates until we reach at the end of our sequence. At the end we are left with state vector a5 that assures us that all inputs <x1, x2, x3, x4, x5> have been fed to our model and an output is generated that is contributed by all outputs.

$$a^{<t>} = s(a^{<t-1>}, x^{<t>})$$

where $a^{<t-1>}$ = contribution from past output

$x^{<t>}$ = Current input

State Vector



Single RNN cell

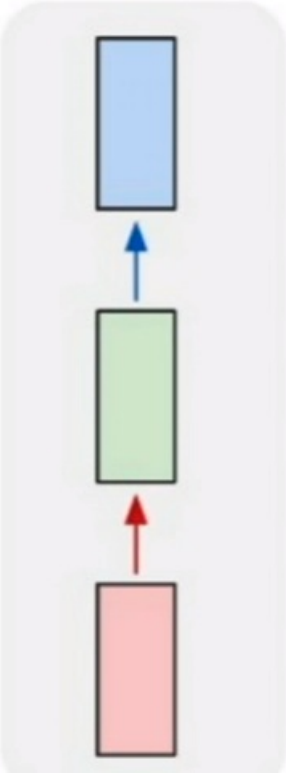
Pseudocode for RNN

```
rnn = RNN()
ff = FeedForwardNN()
hidden_states = [0,0,0,0]

for word in input:
  output, hidden_state = rnn(word, hidden_state)

prediction = ff(output)
```

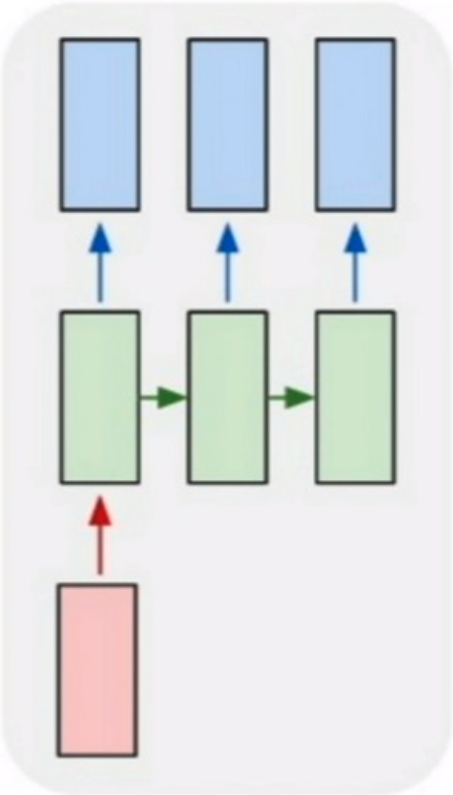
RNN hosted with ❤ by GitHub view raw

Types of RNN architectures

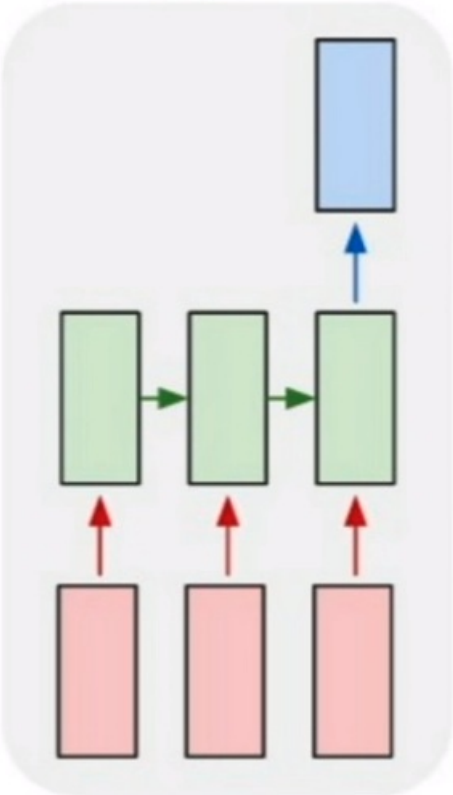
- **One to One**



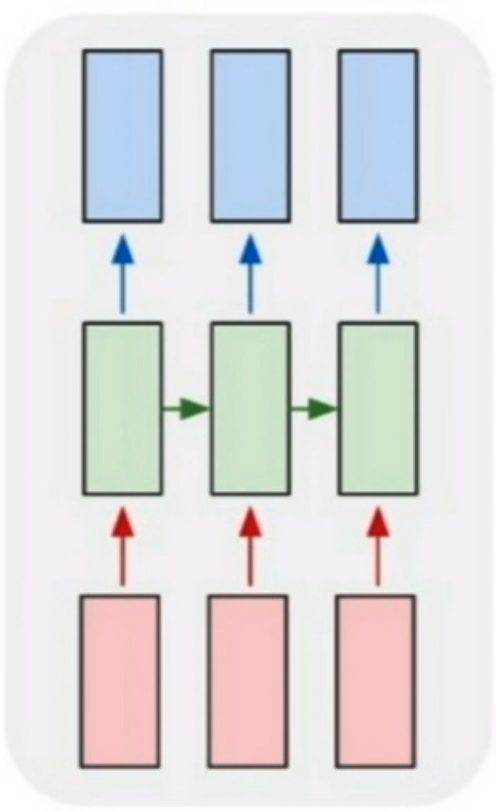
- **One to Many**—These kinds of RNN architectures are usually used for Image captioning/story captioning.



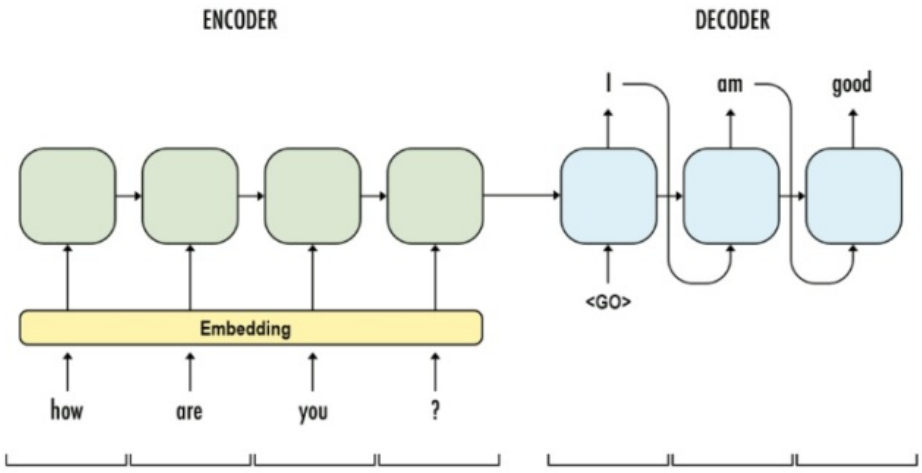
- **Many to One**—These kinds of RNN architectures are used for Sentiment Analysis.



- **Many to Many**—These types of RNN architectures are utilized in Part of Speech i.e. where we are expected to find property for each word.



- **Encoder-Decoder**—These types of RNN are the most complex ones and are used for Language Translation.



Source

Drawbacks of RNN

Short-term Memory

I hope you've pondered upon the odd color distribution in our final RNN cell.

Final output produced by RNN

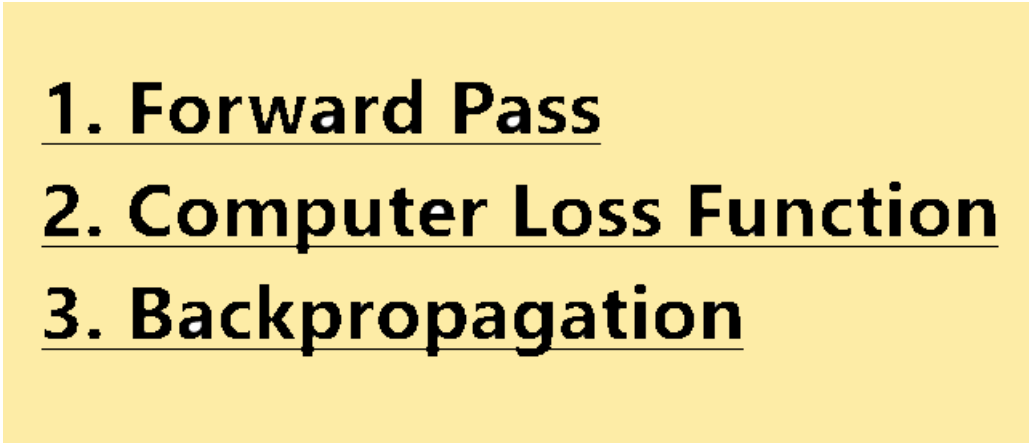
This is an interpretation of Short-term memory. In RNN, at each new timestamp(new input) old information gets morphed by the current input. One could imagine, that after "*t*" timestamps, the information stored at the time step *(t-k)* gets completely morphed.

And thus, RNNs can't be used for very long sequences.

Vanishing Gradient

This is the reason for Short-term memory. Vanishing Gradient is present in every type of Neural Network due to the nature of Backpropagation.

When we train a Neural Network there are three major steps associated with our training. First, a forward pass is done to make a prediction. Later, it compares the prediction to theoretical value producing a loss function. Lastly, we aim to make our prediction better, therefore, we implement Backpropagation that revises values for each node.

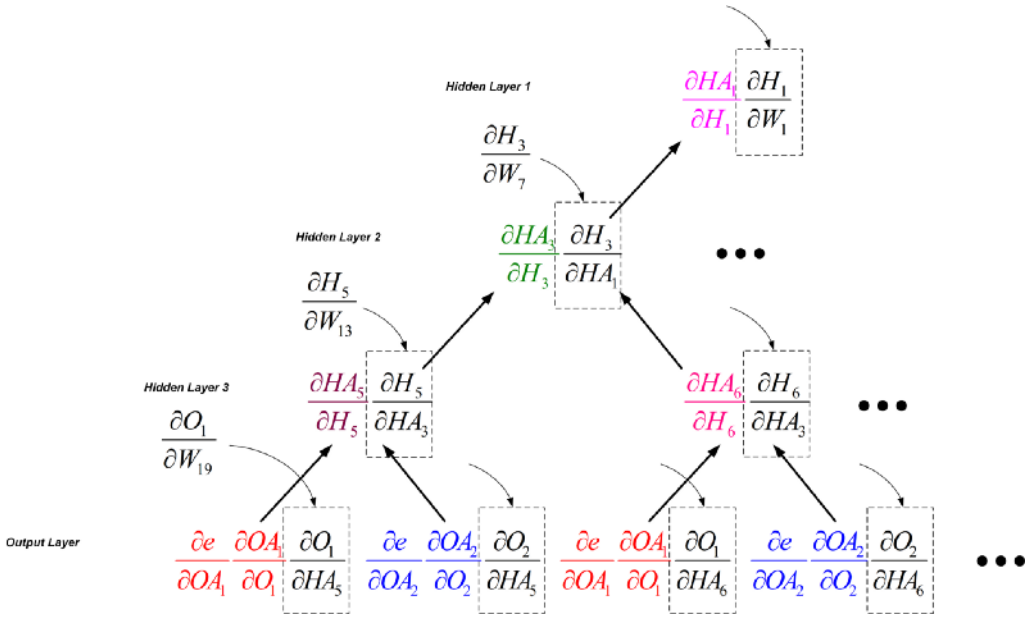


"After calculation of loss function, we're pretty sure that our model is doing something wrong and we need to inspect that, but, it is practically impossible to check for each neuron. But, also the only way possible for us to salvage our model is to retrograde.

Steps for Backpropagation

- We compute certain losses at the output and we will try to figure out which node was responsible for that inefficiency.
- To do so, we will backtrack the whole network.
- Suppose, we found that the second layer(w3h2+b2) is responsible for our loss, and we will try to change it. But if we ponder upon our network, w3 and b2 are independent entities but h2 depends upon w2, b1 & h1 and h1 further depends upon our input i.e. x1, x2, x3...., xn. But since we don't have control over inputs we will try to amend w1 & b1.

To compute our changes we will use the chain rule."



Chain Rule for Backpropagation, Source

When we perform backpropagation, we calculate weights and biases for each node. But, if the improvements in the former layers are meager then the adjustment to the current layer will be much smaller. This causes gradients to dramatically diminish and thus leading to almost NULL changes in our model and due to that our model is no longer learning and no longer improving.

LSTMs and GRUs

To combat the drawbacks of RNNs, we have **LSTM**(*Long Short Term Memory*) and **GRU**(*Gated Recurrent Unit*). LSTMs and GRUs are basically advanced versions of RNNs with little tweaks to overcome the problem of Vanishing Gradients and learning long-term dependencies using components known as "Gates". Gates are a tensor operation that can learn the flow of information and thus short-term memory isn't an issue for them.

- During **Forward propagation**, the gates control flow of information. Thus, preventing any irrelevant information from being written to states.
- During **Backpropagation**, the gates control the flow of gradient, and these gates are capable of multiplying the gradients to avoid vanishing gradient.

To learn more about LSTM and GRUs, you can check out:

Illustrated Guide to LSTM's and GRU's: A step by step explanation
*Hi and welcome to an Illustrated Guide to Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). I'm Michael...*towardsdatascience.com

> LSTM doesn't solve problem of Exploding Gradients, therefore, we tend to use Gradient Clipping while implementing LSTMs.

Conclusion

Hopefully, this article will help you to understand about Recurrent Neural Network in the best possible way and also assist you to its practical usage.

As always, thank you so much for reading, and please share this article if you found it useful!

Feel free to connect:

> *LinkedIn ~ https://www.linkedin.com/in/dakshtrehan/*
>
> *Instagram ~ https://www.instagram.com/_daksh_trehan_/*
>
> *Github ~ https://github.com/dakshtrehan*

Follow for further Machine Learning/ Deep Learning blogs.

> *Medium ~ https://medium.com/@dakshtrehan*

Want to learn more?

Detecting COVID-19 Using Deep Learning

The Inescapable AI Algorithm: TikTok

An insider's guide to Cartoonization using Machine Learning

Why are YOU responsible for George Floyd's Murder and Delhi Communal Riots?

Convolution Neural Network for Dummies

Diving Deep into Deep Learning

Why Choose Random Forest and Not Decision Trees

Clustering: What it is? When to use it?

Start off your ML Journey with k-Nearest Neighbors

Naive Bayes Explained

Activation Functions Explained

Parameter Optimization Explained

Gradient Descent Explained

Logistic Regression Explained

Linear Regression Explained

Determining Perfect Fit for your ML Model

Cheers!

By Daksh Trehan on July 30, 2020.

Canonical link

Exported from Medium on August 2, 2020.