- RNN Architecture -

RNN Introduction:

Recurrent Neural Networks is a class of **Artificial Neural Networks** that distinguishes from other feedforward neural networks by introducing to learn from the **sequential** type of data.

The sequential data consists of a sequence of texts in terms of words that are connected to provide the context and proper meaning. We can think of it as an impeccable English sentence irrespective of the number of words present in it which gives us an insight of meaning the user wants to share with us.

The fully connected feedforward neural networks consist of an input layer, output layer and defined number of hidden layers that need the independent features to learn from them and are not useful to perform the learning on the sequential type of the data. The reason behind that the fully connected networks require an identical number of features to be provided in each iteration. And in the normal sequential types of data if we consider loads of restaurant reviews not all of the sentences deserve the same length of words. This perhaps fails the provision that fully connected neural networks possess and will not be able to execute the computation. Here, the RNN comes into the play.

Forms of RNN:

Before going depth of the RNN functioning let me introduce many styles of RNN where we can use them depending on the problem statement we hold.

- Many-to-Many (Equal Number of Input and Output Sequence Length | Example: Named Entity Recognition)
- Many-to-Many (Variable Number of Input and Output Sequence Length | Example: Neural Machine Translation like English to German)
- Many-to-One (Variable Number of Input and Only One Output | Example: Text Classification or Sentiment Analysis)
- One-to-Many (Only One Input and Variable Number of Output | Example: Music Generation, Image Captioning)

As you can observe above patterns of the RNN different nature of problem statements emerged in the real world we can easily solve.

RNN Working:

Let us dive and learn how the RNN mechanism works. We will try to understand the complete architecture, the mathematics behind that and implement the RNN using Tensorflow. There are three main layers included in the RNN structure namely the input layer where the sequential data ignites, the hidden layer where actual learning starts and the output layer where RNN computes the response.

Input Layer:

As RNN is useful for the sequential data, the input we will sequentially pass to the network i.e. word by word where each word will be transferred over the network one at a time. Here every word is passed in a single individual neuron and thus input layer can be treated as a chain of the neurons connected.

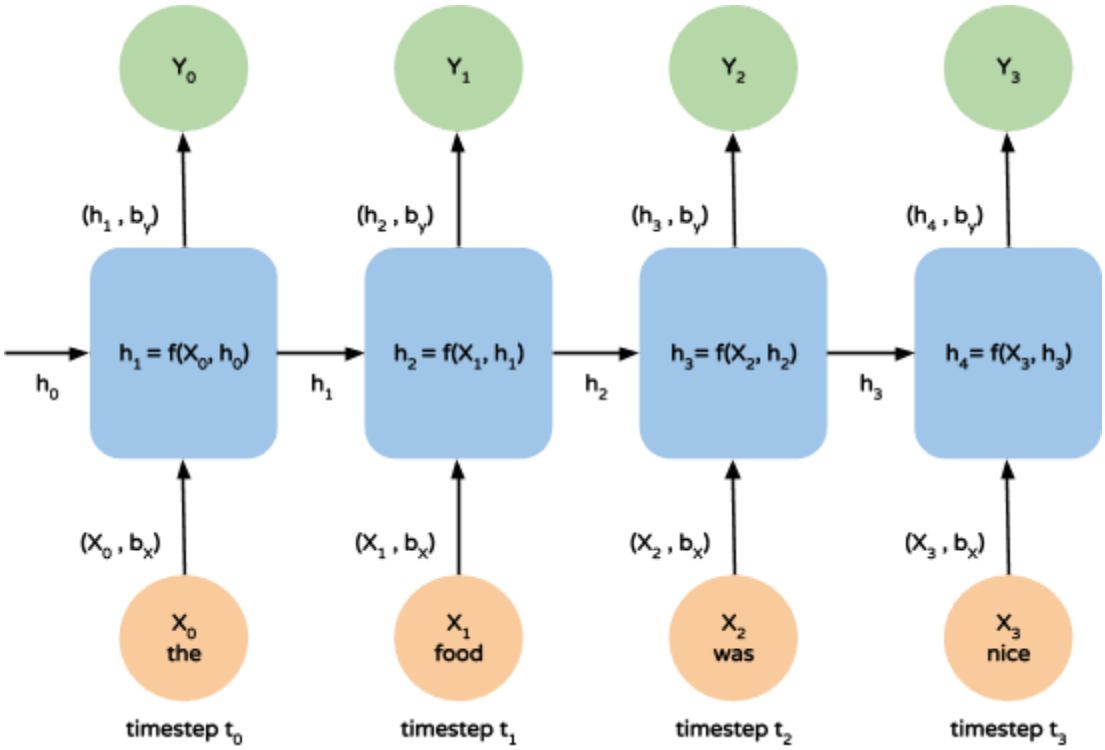
Hidden Layer:

The input goes into the hidden layer where the RNN tries to learn the context and keeps in the memory to further process it. The output of the hidden neuron is passed to the next neuron and the chain goes on. This way the RNN tries to learn the complete context of the sequence.

Output Layer:

The response propagates from the hidden layer and can be determined as the final response or the input to the next layer depending on the form of the RNN used.

Let us consider the restaurant review we have - **"The food was delicious"**
Initially, the first word goes in the neuron (input), reckoning the first input (hidden) and the response generates (output) to process further. We can picturise the workflow as follows:



Mathematics and Computations:
We will go step by step picturized above and gain the understanding of computations involved.

At timestep one **(t_0)**:

- The first word ***"the"*** will pass as an input denoted by X_0
- At first timestep, i.e. t_0, h_0 which is the activation output from the previous timestep, in this case, no previous timestep is present is set as zero
- In the hidden state or neuron, the activation is computed, in RNN usually, we can use the "***tanh" or "ReLu"*** activation function to compute the hidden state output
- Initialized weights WX_0 are passed with the input X_0 and Wh_0 with the previous timestep's hidden state output along with the bias b_X

Following is the equation to derive the first hidden state output h_1

$$h_1 = f(WX_0 * X_0 + Wh_0 * h_0 + b_X)$$

- This calculated hidden output is used to evaluate the output response Y_0 with initialized weights Wh_1 and the bias b_Y

Below is the equation to derive the first output response Y_0

$$Y_0 = f(Wh_1 * h_1 + b_Y)$$

- The function is used can be any activation function to evaluate the hidden output. This hidden output h_1 is passed to the next neuron along with the next input word ***"food"*** i.e. X_1 at the next timestep t_1

A similar computation is conducted at all next timesteps where at each timestep the current input and previous state's output is passed to compute the current state's hidden output and produce the output response.

One thing to note here, at each timestep all the initialized weights are shared across all neurons.

The above-described process is the very basic flow to understand how the RNN works and the computations vary with the problem statement we are trying to solve.

Backpropagation in RNN:

As many of us know in the artificial neural network the most important part is to adjust the weights which we assigned to the features randomly during each iteration. The concept is termed as backpropagation.

These weights determine how much importance should we give to each of the features. In the due process, we calculate gradients concerning the loss or error generated at the output. The most popular method to calculate the gradients is the Gradient Descent technique.

In case of the RNN, as we pass the inputs sequentially over the timesteps in the forward propagation, at the last timestep the error or loss is calculated, this loss is propagated back to adjust the weights assigned at each timestep and the complete process is termed as **Backpropagation through the time (BPTT)**.

The gradients are calculated by deriving the derivatives of the current timesteps weights concerning the loss.

In this fashion, the RNN works. This is a very basic form of the RNN and also is termed as **Vanilla RNN**.