Week 09: RNN

Table of Contents

Need for a Neural Network dealing with Sequences

Before we deep dive into the details of what a recurrent neural network is, let's ponder a bit on if we really need a network specially for dealing with sequences in information. Also what are the kinds of tasks that we can achieve using such networks?

The beauty of recurrent neural networks lies in their diversity of application. When we are dealing with RNNs they have a great ability to deal with various input and output types.

Sentiment Classification – This can be a task of simply classifying tweets into positive and negative sentiment. So here the input would be a tweet of varying lengths, while output is of a fixed type and size.

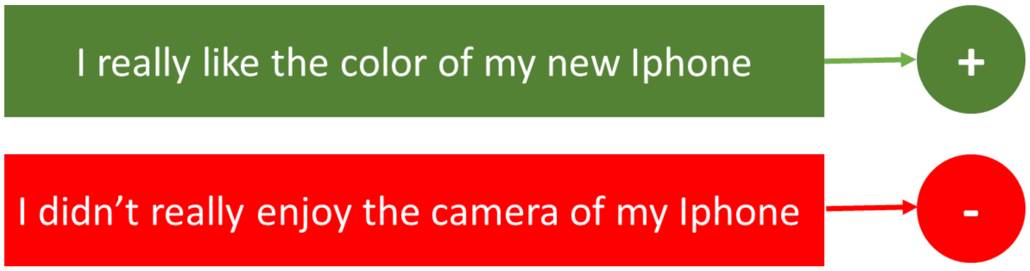

I really like the color of my new Iphone
+
I didn't really enjoy the camera of my Iphone
-

Image Captioning – Here, let's say we have an image for which we need a textual description. So we have a single input – the image, and a series or sequence of words as output. Here the image might be of a fixed size, but the output is a description of varying lengths

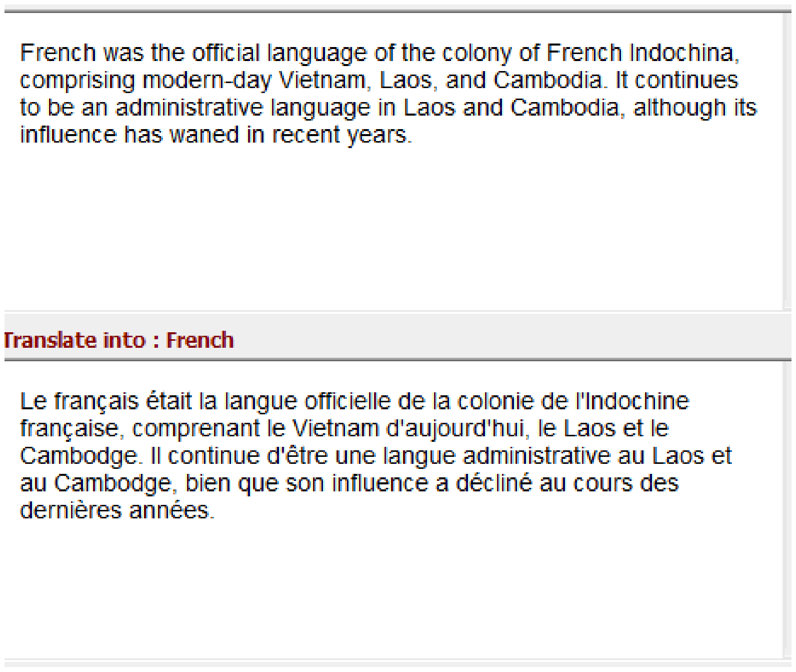
A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

- **Language Translation** – This basically means that we have some text in a particular language, let's say English, and we wish to translate it into French. Each language has its own semantics and would have varying lengths for the same sentence. So here the inputs as well as outputs are of varying lengths.


French was the official language of the colony of French Indochina, comprising modern-day Vietnam, Laos, and Cambodia. It continues to be an administrative language in Laos and Cambodia, although its influence has waned in recent years.

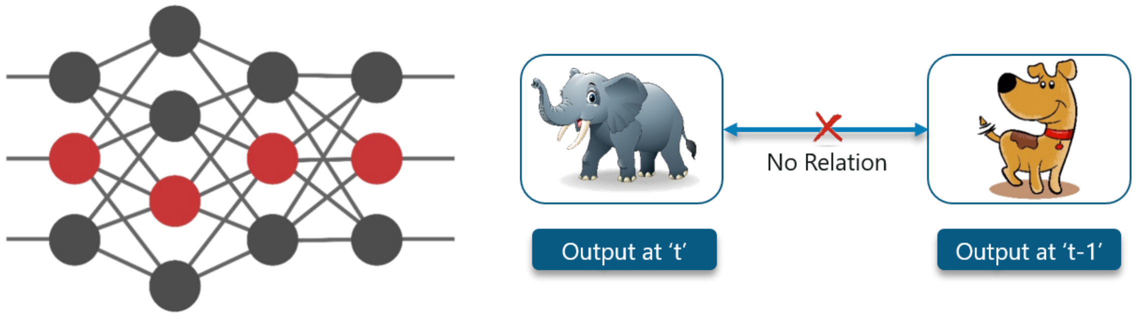
Le français était la langue officielle de la colonie de l'Indochine française, comprenant le Vietnam d'aujourd'hui, le Laos et le Cambodge. Il continue d'être une langue administrative au Laos et au Cambodge, bien que son influence a décliné au cours des dernières années.

So RNNs can be used for mapping inputs to outputs of varying types, lengths and are fairly generalized in their application. Looking at their applications, let’s see how the architecture of an RNN looks like.

Why Not Feedforward Networks?

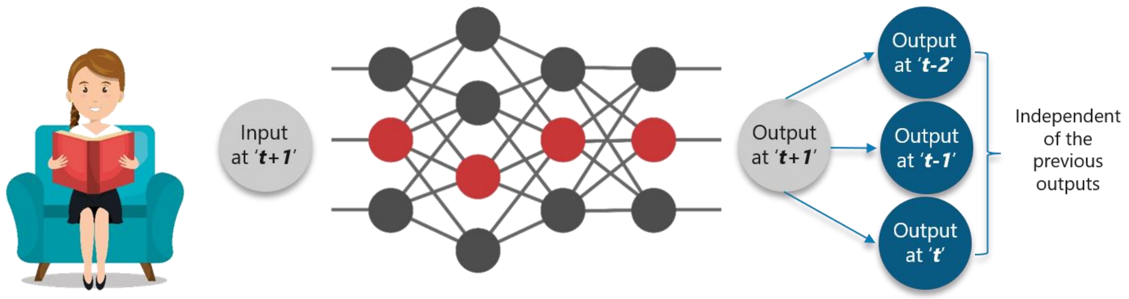
Consider an image classification use-case where you have trained the neural network to classify images of various animals. So, let’s say you feed in an image of a cat or a dog, the network actually provides an output with a corresponding label to the image of a cat or a dog respectively.

Consider the following diagram:



Here, the first output being an elephant will have no influence on the previous output which was a dog. This means that output at time ‘t’ is independent of output at time ‘t-1’.

Consider this scenario where you will require the use of the previously obtained output:

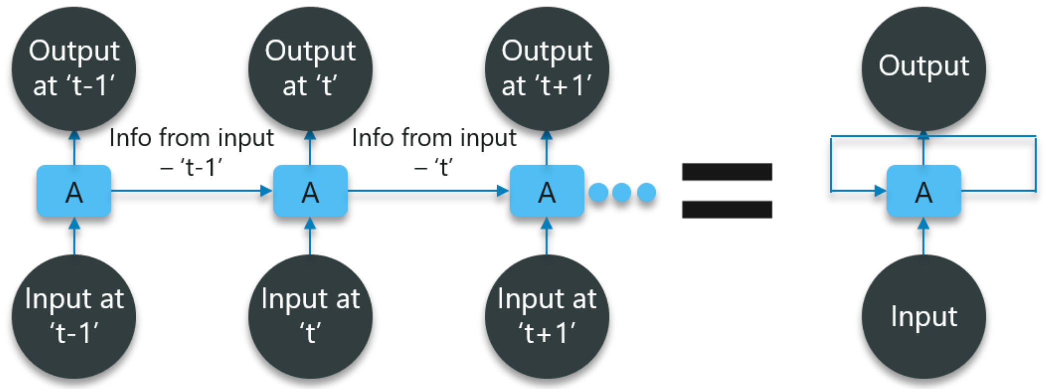


The concept is similar to reading a book. With every page you move forward into, you need the understanding of the previous pages to make complete sense of the information ahead in most of the cases.

With a feed-forward network the new output at time ‘t+1’ has no relation with outputs at either time t, t-1 or t-2. So, feed-forward networks cannot be used when predicting a word in a sentence as it will have no absolute relation with the previous set of words.

But, with Recurrent Neural Networks, this challenge can be overcome.

Consider the following diagram:



In the above diagram, we have certain inputs at 't-1' which are fed into the network. These inputs will lead to corresponding outputs at time 't-1' as well.

At the next timestamp, information from the previous input 't-1' is provided along with the input at 't' to eventually provide the output at 't' as well.

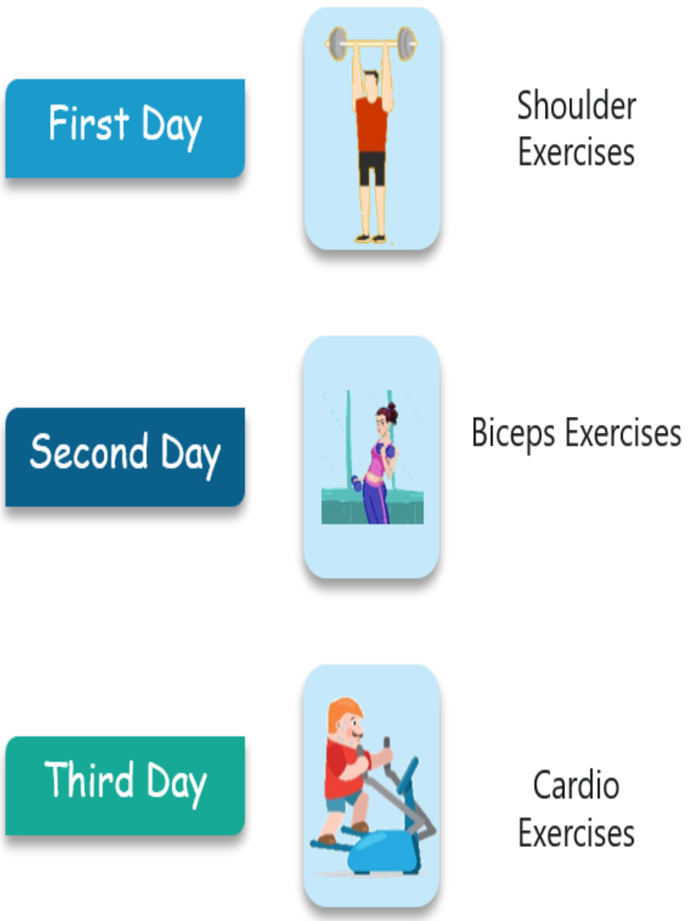
This process repeats, to ensure that the latest inputs are aware and can use the information from the previous timestamp.

Next up in this Recurrent Neural Networks blog, we need to check out what Recurrent Neural Networks (RNNs) actually are.

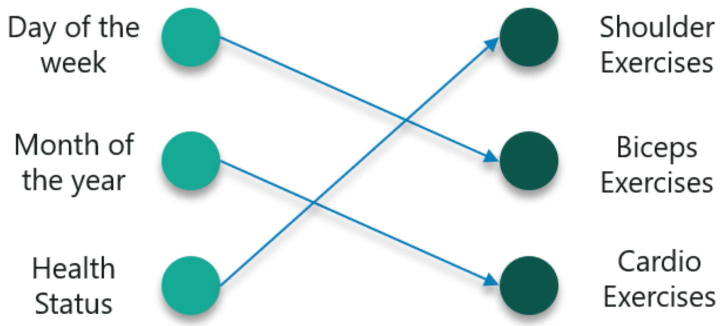
What Are Recurrent Neural Networks?

Recurrent Networks are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, numerical times series data emanating from sensors, stock markets and government agencies.

For a better clarity, **consider the following analogy:** You go to the gym regularly and the trainer has given you the following schedule for your workout:

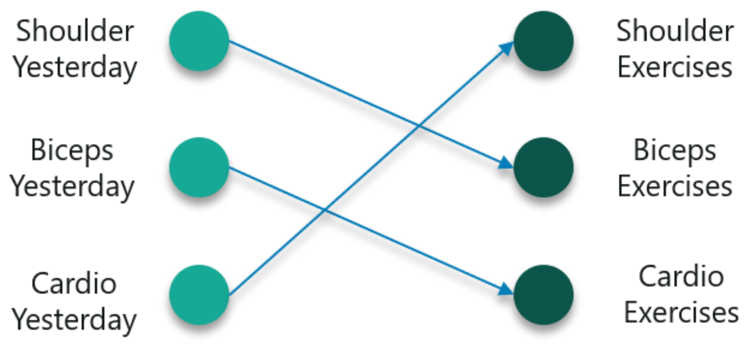


Note that all these exercises are repeated in a proper order every week. First, let us use a feed-forward network to try and predict the type of exercise.



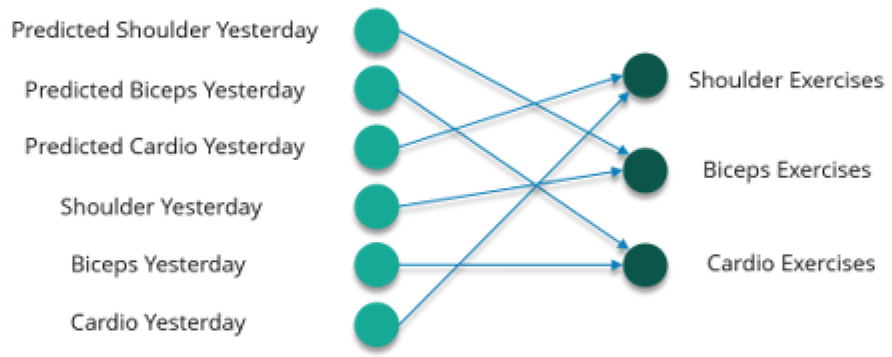
The inputs are day, month and health status. A neural network has to be trained using these inputs to provide us with the prediction of the exercises.

However, this will not be very accurate considering the input. To fix this, we can make use of the concept of Recurrent Neural Networks as shown below:

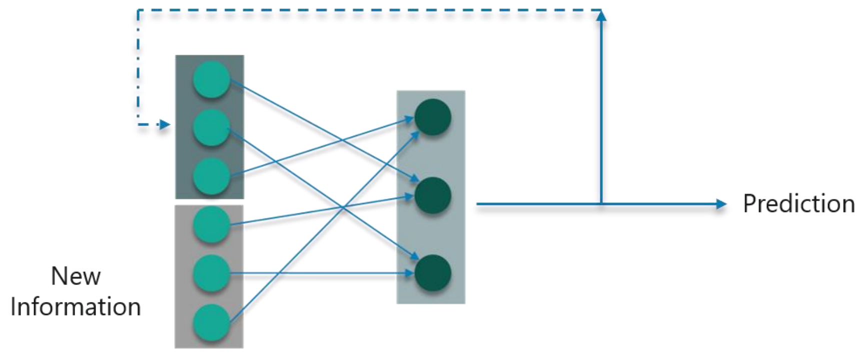


In this case, consider the inputs to be the workout done on the previous day. So if you did a shoulder workout yesterday, you can do a bicep exercise today and this goes on for the rest of the week as well.

However, if you happen to miss a day at the gym, the data from the previously attended timestamp can be considered as shown below:

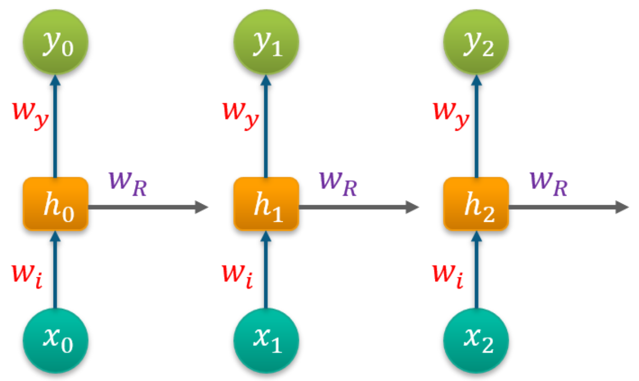


If a model is trained based on the data it can obtain from the previous exercises, the output from the model will be extremely accurate. To sum it up, let us convert the data we have into vectors. Well, what are vectors? Vectors are numbers which are input to the model to denote if you have done the exercise or not.



So, if you have a shoulder exercise, the corresponding node will be ‘1’ and the rest of the exercise nodes will be mapped to ‘0’.

Let us check out the math behind the working of the neural network. Consider the following diagram:



$$h^{(t)} = g_h\left(w_i x^{(t)} + w_R h^{(t-1)} + b_h\right)$$

$$y^{(t)} = g_y\left(w_y h^{(t)} + b_y\right)$$

Consider 'w' to be the weight matrix and 'b' being the bias:

h1 = tanh(wi*x1+wR*h0+b)

y1 = tanh(h1*wy+b)

h2 = tanh(wi*x2+h1*wR*+b)

Y2 = tanh(hi*wy+b)

At time t=0, input is 'x0' and the task is to figure out what is 'h0'. Substituting t=0 in the equation and obtaining the function h(t) value. Next, the value of 'y0' is found out using the previously calculated values when applied to the new formula.

This process is repeated through all of the timestamps in the model to train a model.

What is a Recurrent Neural Network (RNN)?

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer. Below is how you can convert a Feed-Forward Neural Network into a Recurrent Neural Network:

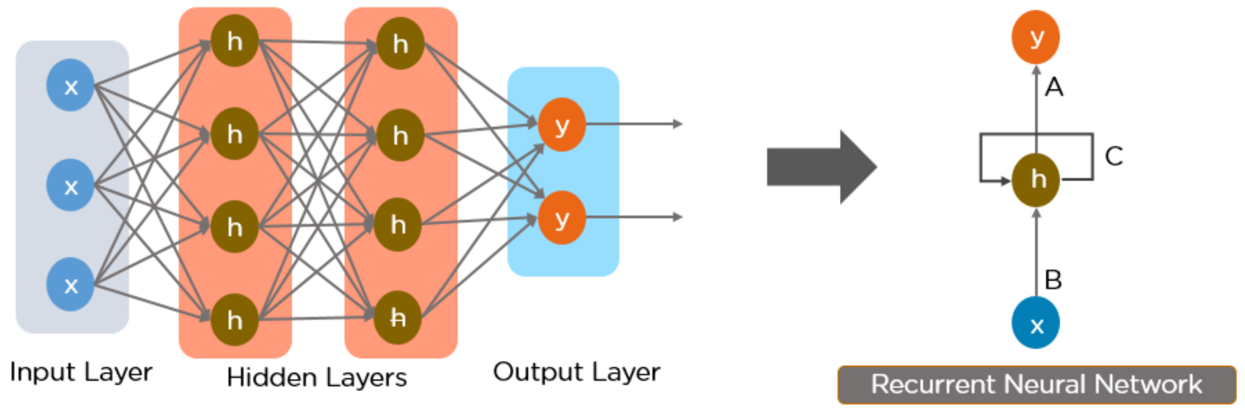


Fig: Simple Recurrent Neural Network

The nodes in different layers of the neural network are compressed to form a single layer of recurrent neural networks. A, B, and C are the parameters of the network.

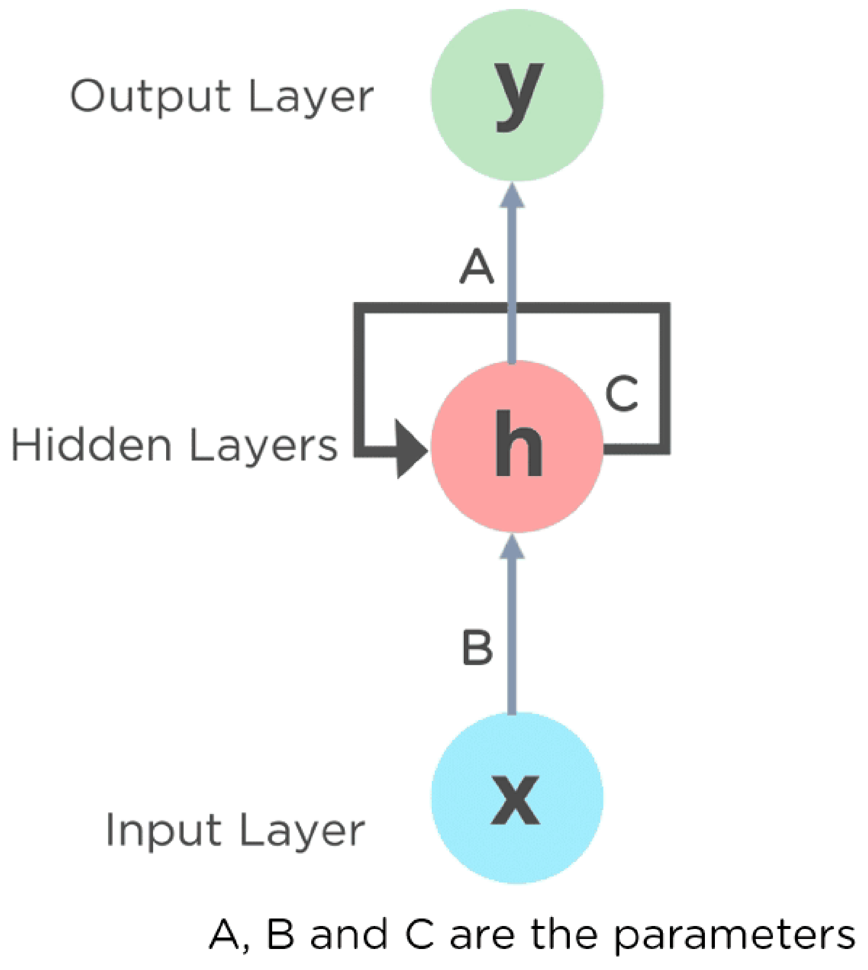
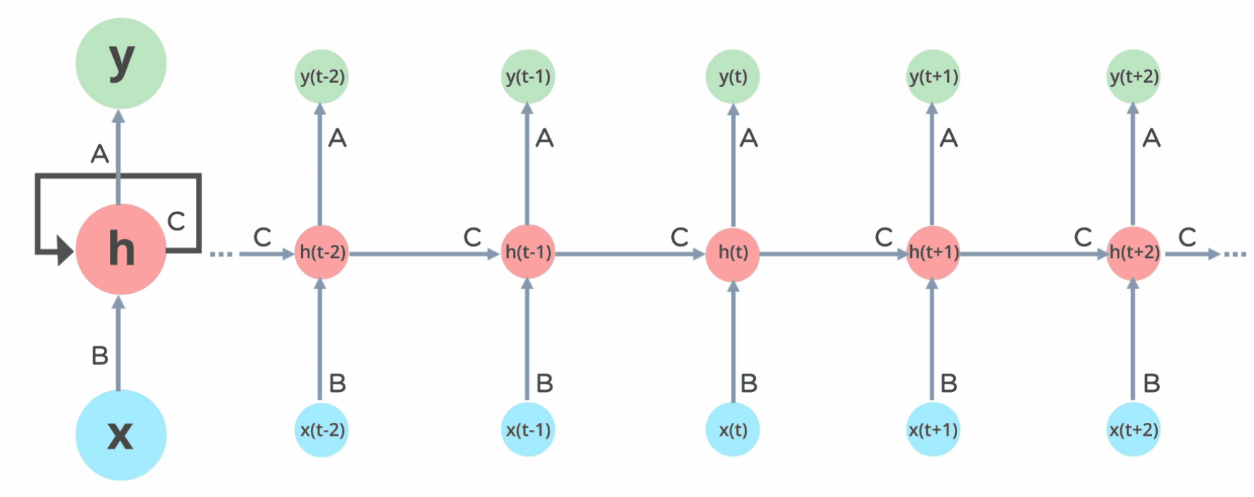


Fig: Fully connected Recurrent Neural Network



Here, “x” is the input layer, “h” is the hidden layer, and “y” is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.

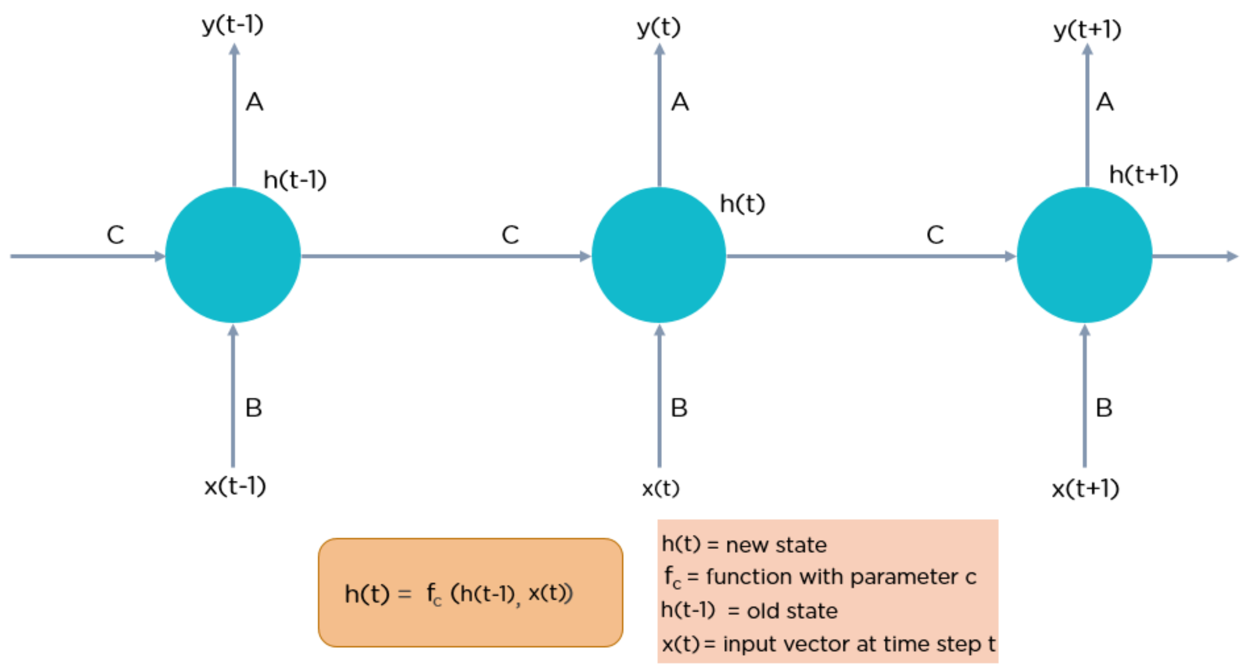


Fig: Fully connected Recurrent Neural Network

Now that you understand what a recurrent neural network is let’s look at the different types of recurrent neural networks.

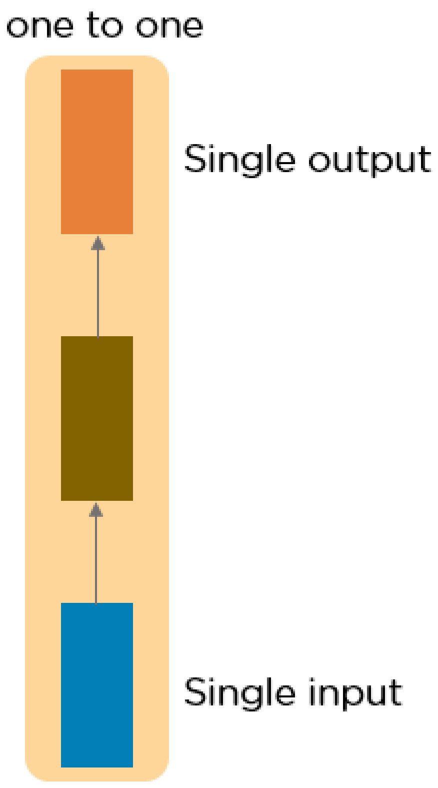
Types of Recurrent Neural Networks

There are four types of Recurrent Neural Networks:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

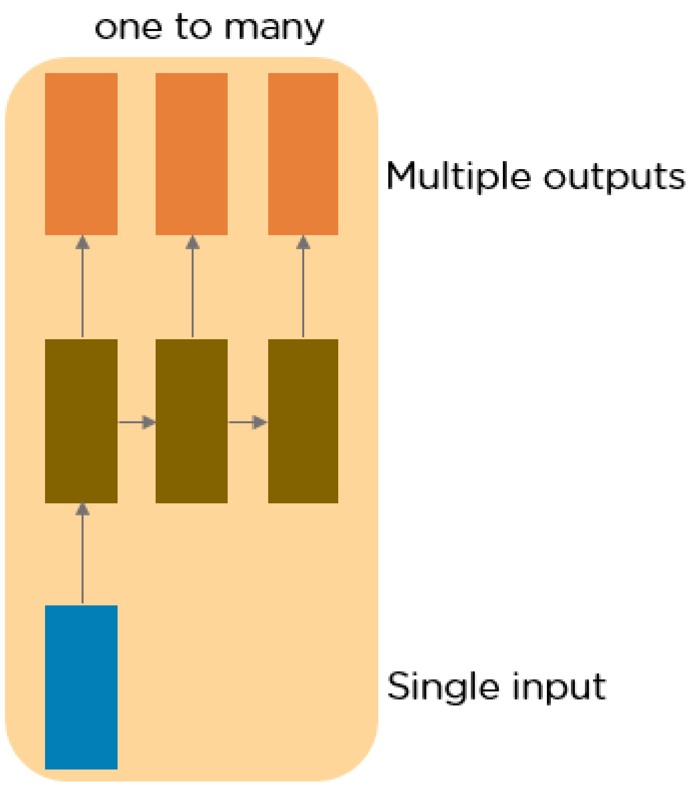
One to One RNN

This type of neural network is known as the Vanilla Neural Network. It's used for general machine learning problems, which has a single input and a single output.



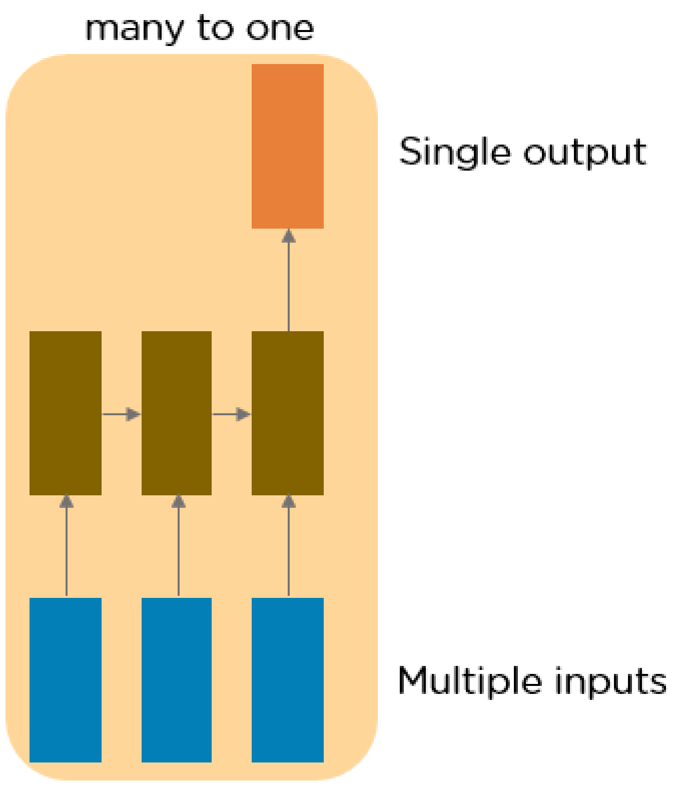
One to Many RNN

This type of neural network has a single input and multiple outputs. An example of this is the image caption.



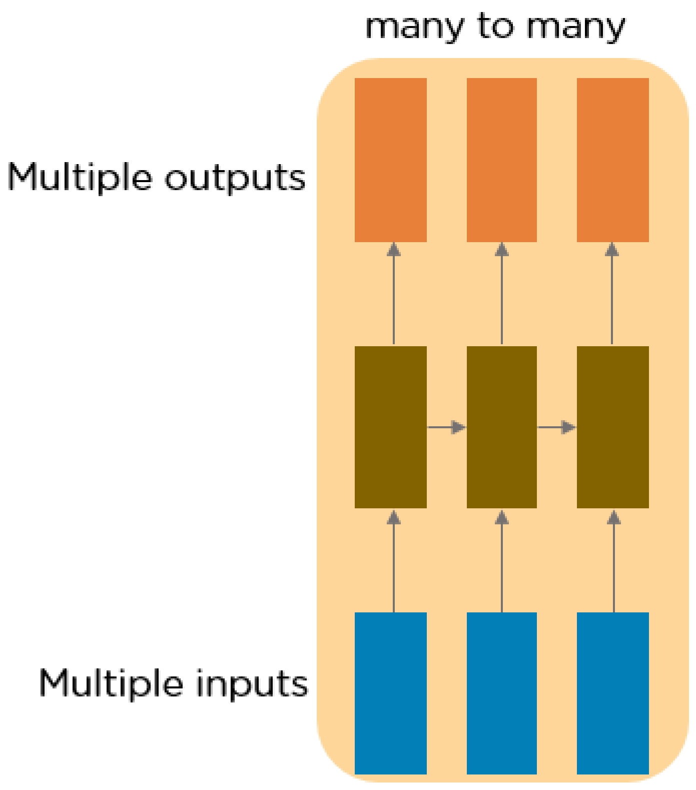
Many to One RNN

This RNN takes a sequence of inputs and generates a single output. Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.



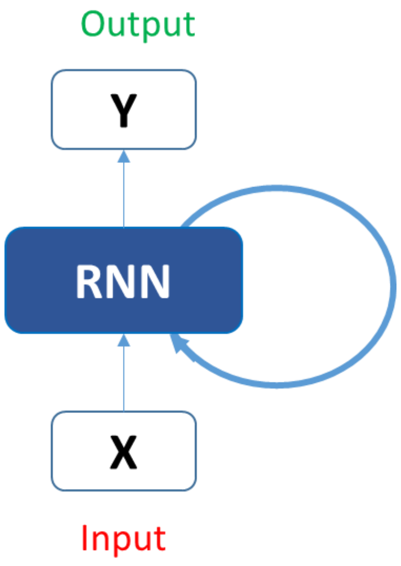
Many to Many RNN

This RNN takes a sequence of inputs and generates a sequence of outputs. Machine translation is one of the examples.

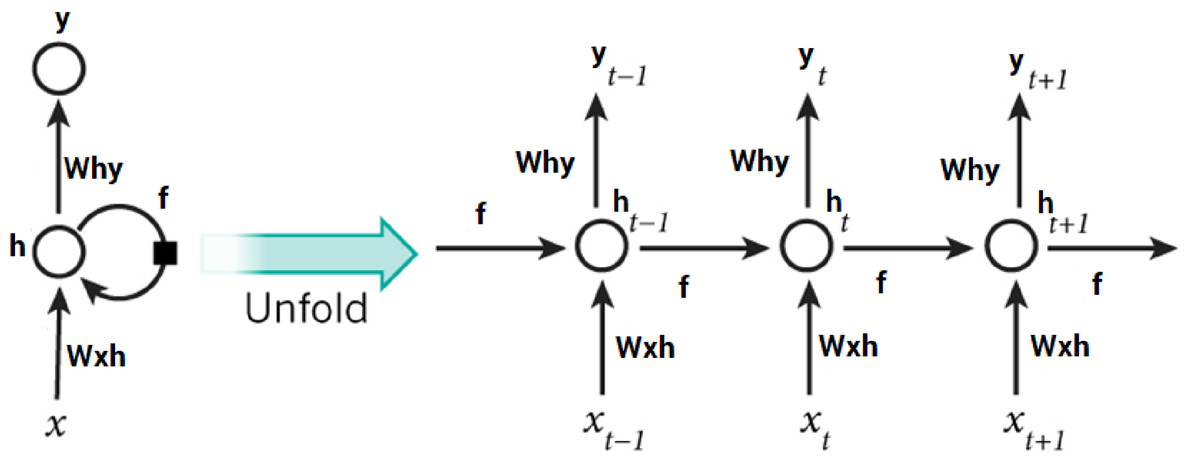


Understanding a Recurrent Neuron in Detail

Let’s take a simple task first. Let’s take a character level RNN where we have a word “Hello”. So we provide the first 4 letters i.e. h,e,l,l and ask the network to predict the last letter i.e.’o’. So here the vocabulary of the task is just 4 letters {h,e,l,o}. In real case scenarios involving natural language processing, the vocabularies include the words in the entire wikipedia database, or all the words in a language. Here for simplicity we have taken a very small set of vocabulary.



Let’s see how the above structure can be used to predict the fifth letter in the word “hello”. In the above structure, the blue RNN block, applies something called a recurrence formula to the input vector and also its previous state. In this case, the letter “h” has nothing preceding it, let’s take the letter “e”. So at the time the letter “e” is supplied to the network, a recurrence formula is applied to the letter “e” and the previous state which is the letter “h”. These are known as various time steps of the input. So if at time t, the input is “e”, at time t-1, the input is “h”. The recurrence formula is applied to e and h both. and we get a new state.



The formula for the current state can be written as –

$$h_t = f(h_{t-1}, x_t)$$

Here, Ht is the new state, ht-1 is the previous state while xt is the current input. We now have a state of the previous input instead of the input itself, because the input neuron would have applied the transformations on our previous input. So each successive input is called a time step.

In this case we have four inputs to be given to the network, during a recurrence formula, the same function and the same weights are applied to the network at each time step.

Taking the simplest form of a recurrent neural network, let's say that the activation function is tanh, the weight at the recurrent neuron is Whh and the weight at the input neuron is Wxh, we can write the equation for the state at time t as –

$$h_t = \tanh\,(W_{hh}h_{t-1} + W_{xh}x_t)$$

The Recurrent neuron in this case is just taking the immediate previous state into consideration. For longer sequences the equation can involve multiple such states. Once the final state is calculated we can go on to produce the output

Now, once the current state is calculated we can calculate the output state as-

$$y_t = W_{hy}h_t$$

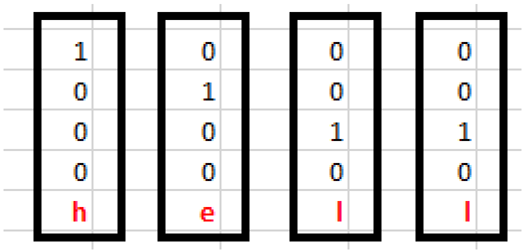
Let me summarize the steps in a recurrent neuron for you-

1. A single time step of the input is supplied to the network i.e. xt is supplied to the network
2. We then calculate its current state using a combination of the current input and the previous state i.e. we calculate ht
3. The current ht becomes ht-1 for the next time step
4. We can go as many time steps as the problem demands and combine the information from all the previous states
5. Once all the time steps are completed the final current state is used to calculate the output yt
6. The output is then compared to the actual output and the error is generated
7. The error is then back propagated to the network to update the weights(we shall go into the details of backpropagation in further sections) and the network is trained

Let's take a look at how we can calculate these states in Excel and get the output.

Forward Propagation in a Recurrent Neuron in Excel

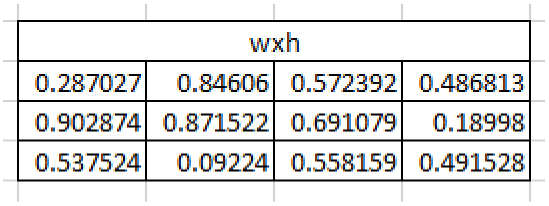
Let’s take a look at the inputs first –



1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	l

The inputs are one hot encoded. Our entire vocabulary is {h,e,l,o} and hence we can easily one hot encode the inputs.

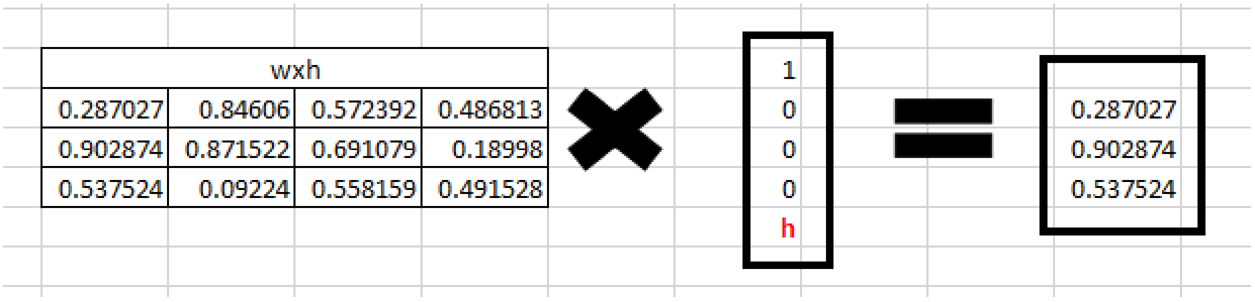
Now the input neuron would transform the input to the hidden state using the weight wxh. We have randomly initialized the weights as a 3*4 matrix –



wxh			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

Step 1:

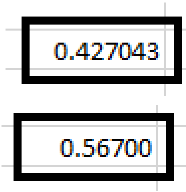
Now for the letter “h”, for the the hidden state we would need Wxh*Xt. By matrix multiplication, we get it as –



$$\begin{bmatrix} 0.287027 & 0.84606 & 0.572392 & 0.486813 \\ 0.902874 & 0.871522 & 0.691079 & 0.18998 \\ 0.537524 & 0.09224 & 0.558159 & 0.491528 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.287027 \\ 0.902874 \\ 0.537524 \end{bmatrix}$$

Step 2:

Now moving to the recurrent neuron, we have Whh as the weight which is a 1*1 matrix and the bias which is also a 1*1 matrix =

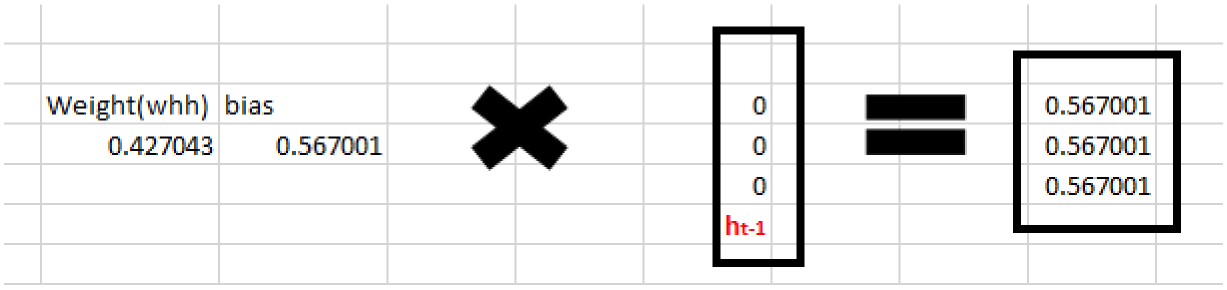


0.427043

0.56700

For the letter “h”, the previous state is [0,0,0] since there is no letter prior to it.

So to calculate -> (whh*ht-1+bias)



Weight(whh)	bias
0.427043	0.567001

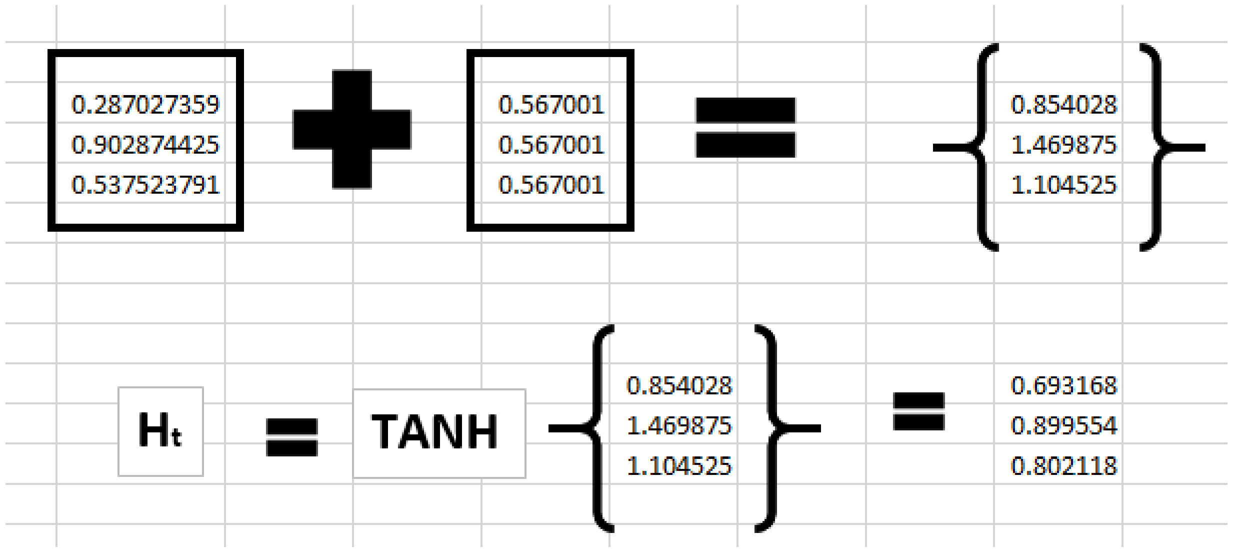
$$\times \begin{bmatrix}0\\0\\0\end{bmatrix}_{h_{t-1}} = \begin{bmatrix}0.567001\\0.567001\\0.567001\end{bmatrix}$$

Step 3:

Now we can get the current state as –

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right)$$

Since for h, there is no previous hidden state we apply the tanh function to this output and get the current state –



$$\begin{bmatrix}0.287027359\\0.902874425\\0.537523791\end{bmatrix} + \begin{bmatrix}0.567001\\0.567001\\0.567001\end{bmatrix} = \left\{\begin{matrix}0.854028\\1.469875\\1.104525\end{matrix}\right\}$$

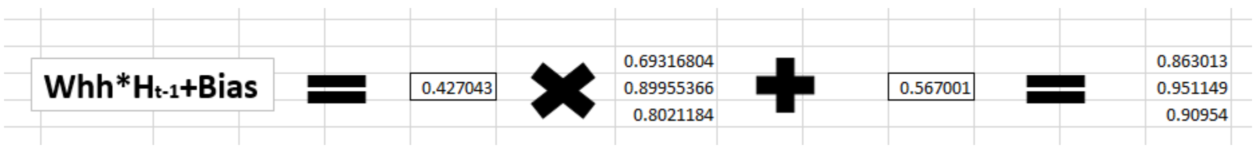
$$\boxed{H_t} = \boxed{\text{TANH}}\left\{\begin{matrix}0.854028\\1.469875\\1.104525\end{matrix}\right\} = \begin{matrix}0.693168\\0.899554\\0.802118\end{matrix}$$

Step 4:

Now we go on to the next state. “e” is now supplied to the network. The processed output of ht, now becomes ht-1, while the one hot encoded e, is xt. Let’s now calculate the current state ht.

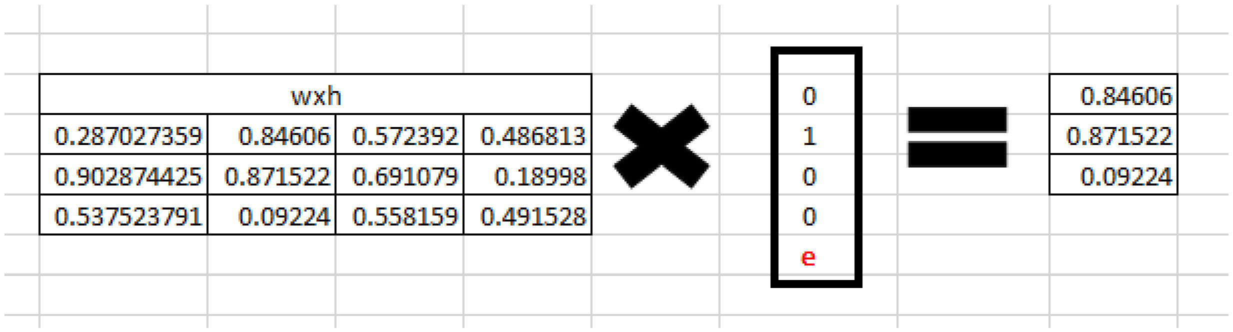
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Whh*ht-1 +bias will be –



$$\boxed{Whh*H_{t-1}+Bias} = \boxed{0.427043} \times \begin{matrix} 0.69316804 \\ 0.89955366 \\ 0.8021184 \end{matrix} + \boxed{0.567001} = \begin{matrix} 0.863013 \\ 0.951149 \\ 0.90954 \end{matrix}$$

Wxh*xt will be –

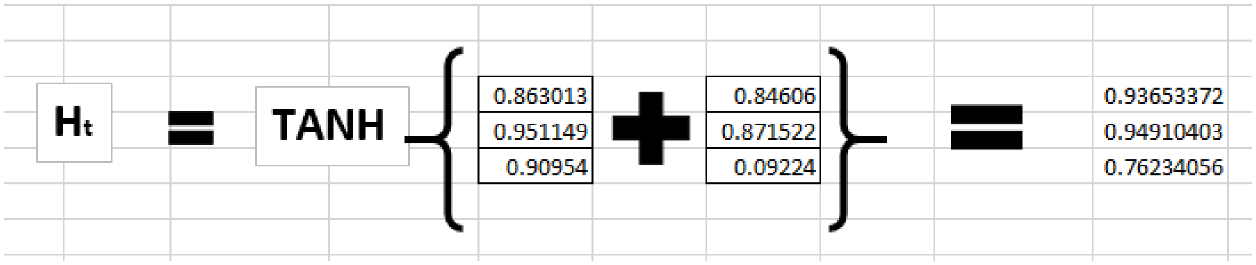


wxh			
0.287027359	0.84606	0.572392	0.486813
0.902874425	0.871522	0.691079	0.18998
0.537523791	0.09224	0.558159	0.491528

$$\times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ e \end{bmatrix} = \begin{bmatrix} 0.84606 \\ 0.871522 \\ 0.09224 \end{bmatrix}$$

Step 5:

Now calculating ht for the letter “e”,



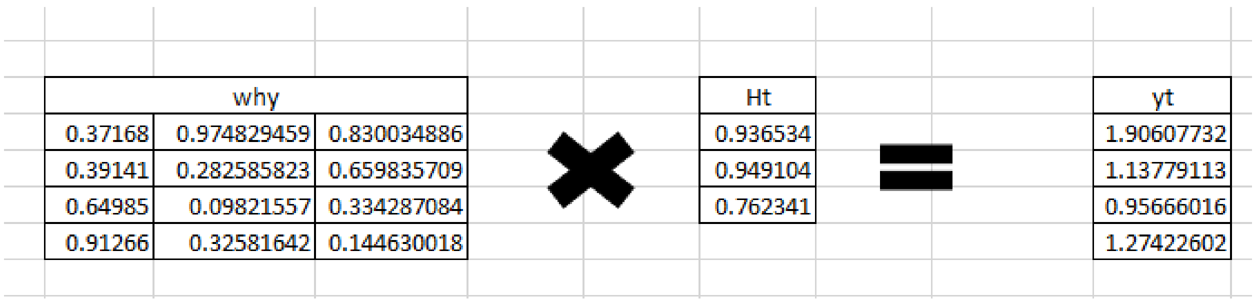
$$\boxed{H_t} = \boxed{TANH} \left\{ \begin{bmatrix} 0.863013 \\ 0.951149 \\ 0.90954 \end{bmatrix} + \begin{bmatrix} 0.84606 \\ 0.871522 \\ 0.09224 \end{bmatrix} \right\} = \begin{matrix} 0.93653372 \\ 0.94910403 \\ 0.76234056 \end{matrix}$$

Now this would become ht-1 for the next state and the recurrent neuron would use this along with the new character to predict the next one.

Step 6:

At each state, the recurrent neural network would produce the output as well. Let’s calculate yt for the letter e.

$$y_t = W_{hy}h_t$$



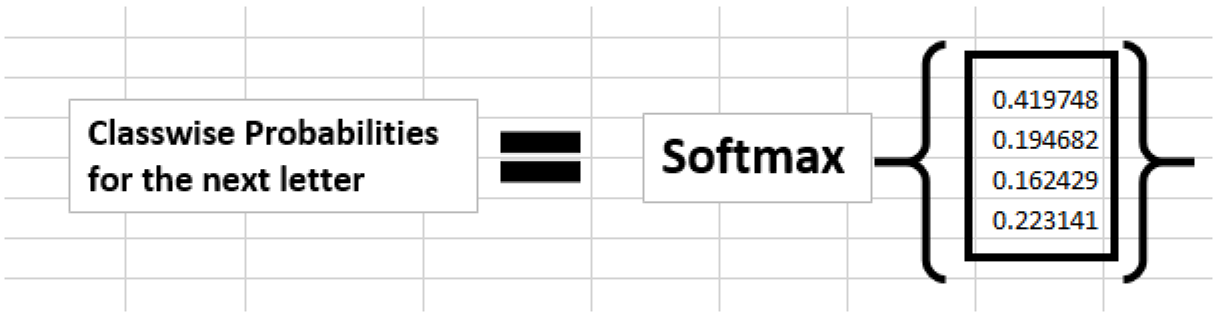
why		
0.37168	0.974829459	0.830034886
0.39141	0.282585823	0.659835709
0.64985	0.09821557	0.334287084
0.91266	0.32581642	0.144630018

Ht
0.936534
0.949104
0.762341

yt
1.90607732
1.13779113
0.95666016
1.27422602

Step 7:

The probability for a particular letter from the vocabulary can be calculated by applying the softmax function. so we shall have softmax(yt)



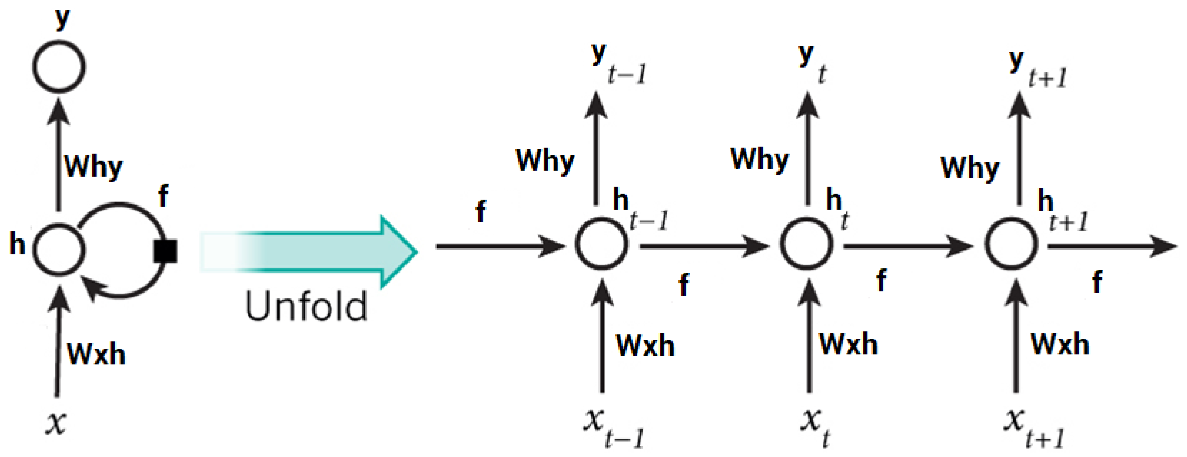
If we convert these probabilities to understand the prediction, we see that the model says that the letter after “e” should be h, since the highest probability is for the letter “h”. Does this mean we have done something wrong? No, so here we have hardly trained the network. We have just shown it two letters. So it pretty much hasn’t learnt anything yet.

Now the next BIG question that faces us is how does Backpropagation work in case of a Recurrent Neural Network. How are the weights updated while there is a feedback loop?

Back propagation in a Recurrent Neural Network(BPTT)

To imagine how weights would be updated in case of a recurrent neural network, might be a bit of a challenge. So to understand and visualize the back propagation, let’s unroll the network at all the time steps. In an RNN we may or may not have outputs at each time step.

In case of a forward propagation, the inputs enter and move forward at each time step. In case of a backward propagation in this case, we are figuratively going back in time to change the weights, hence we call it the Backpropagation through time(BPTT).



In case of an RNN, if yt is the predicted value ȳt is the actual value, the error is calculated as a cross entropy loss –

Et(ȳt,yt) = – ȳt log(yt)

E(ȳ,y) = – ∑ ȳt log(yt)

We typically treat the full sequence (word) as one training example, so the total error is just the sum of the errors at each time step (character). The weights as we can see are the same at each time step. Let’s summarize the steps for backpropagation

1. The cross entropy error is first computed using the current output and the actual output
2. Remember that the network is unrolled for all the time steps
3. For the unrolled network, the gradient is calculated for each time step with respect to the weight parameter
4. Now that the weight is the same for all the time steps the gradients can be combined together for all time steps
5. The weights are then updated for both recurrent neuron and the dense layers

The unrolled network looks much like a regular neural network. And the back propagation algorithm is similar to a regular neural network, just that we combine the gradients of the error for all time steps. Now what do you think might happen, if there are 100s of time steps. This would basically take really long for the network to converge since after unrolling the network becomes really huge.

In case you do not wish to deep dive into the math of backpropagation, all you need to understand is that back propagation through time works similar as it does in a regular neural network once you unroll the recurrent neuron in your network. H

Training Recurrent Neural Networks

Recurrent Neural Networks use a backpropagation algorithm for training, but it is applied for every timestamp. It is commonly known as Back-propagation Through Time (BTT).

There are some issues with Back-propagation such as:

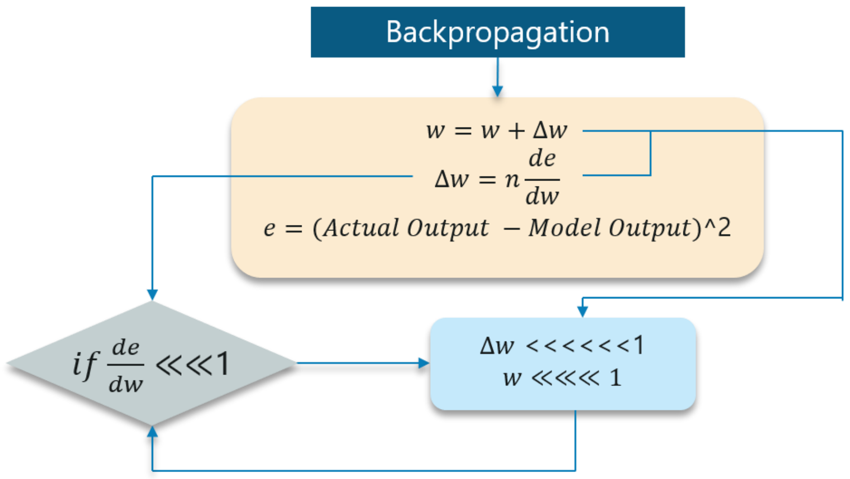
- Vanishing Gradient
- Exploding Gradient

Let us consider each of these to understand what is going on

Vanishing Gradient

When making use of back-propagation the goal is to calculate the error which is actually found out by finding out the difference between the actual output and the model output and raising that to a power of 2.

Consider the following diagram:


Backpropagation
$w = w + \Delta w$
$\Delta w = n \frac{de}{dw}$
$e = (Actual\ Output\ - Model\ Output)$^2
$if \frac{de}{dw} \lll 1$
$\Delta w <<<<<< 1$
$w \lll\lll 1$

With the error calculated, the changes in the error with respect to the change in the weight is calculated. But with each learning rate, this has to be multiplied with the same.

So, the product of the learning rate with the change leads to the value which is the actual change in the weight.

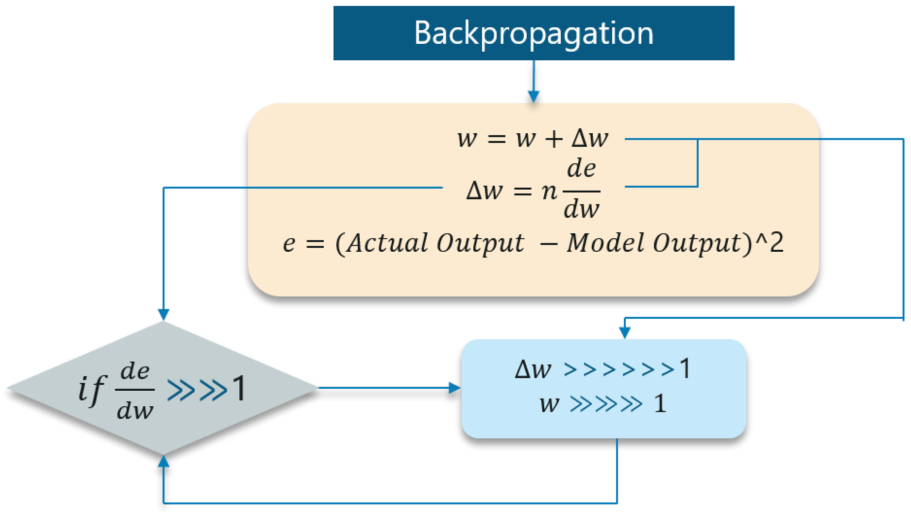
This change in weight is added to the old set of weights for every training iteration as shown in the figure. The issue here is when the change in weight is multiplied, the value is very less.

Consider you are predicting a sentence say,“I am going to France” and you want to predict “I am going to France, the language spoken there is _____”

A lot of iterations will cause the new weights to be extremely negligible and this leads to the weights not being updated.

Exploding Gradient

The working of the exploding gradient is similar but the weights here change drastically instead of negligible change. Notice the small change in the diagram below:



Backpropagation
$w = w + \Delta w$
$\Delta w = n\frac{de}{dw}$
$e = (Actual\ Output\ - Model\ Output)$^2
$if\ \frac{de}{dw} \gg\gg 1$
$\Delta w >>>>>> 1$
$w \gg\gg 1$

References

https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

https://www.edureka.co/blog/recurrent-neural-networks/

Recurrent Neural Network | Fundamentals Of Deep Learning

How do Convolutional Neural Networks work? (e2eml.school)

https://www.youtube.com/watch?v=zD13uQIgac8&ab_channel=edureka%21

https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

Lab Task

https://www.analyticsvidhya.com/blog/2019/01/fundamentals-deep-learning-recurrent-neural-networks-scratch-python/