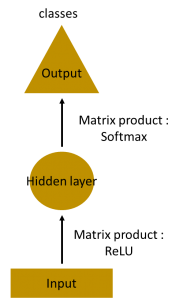
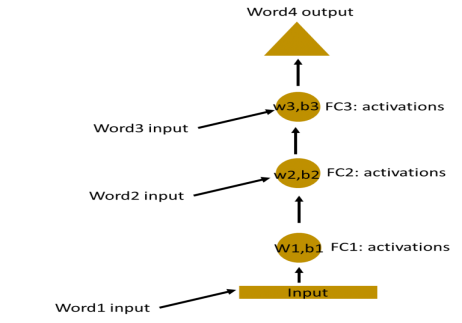
What are Recurrent Neural Networks?

Let’s say the task is to predict the next word in a sentence. Let’s try accomplishing it using an MLP. So what happens in an MLP. In the simplest form, we have an input layer, a hidden layer and an output layer. The input layer receives the input, the hidden layer activations are applied and then we finally receive the output.

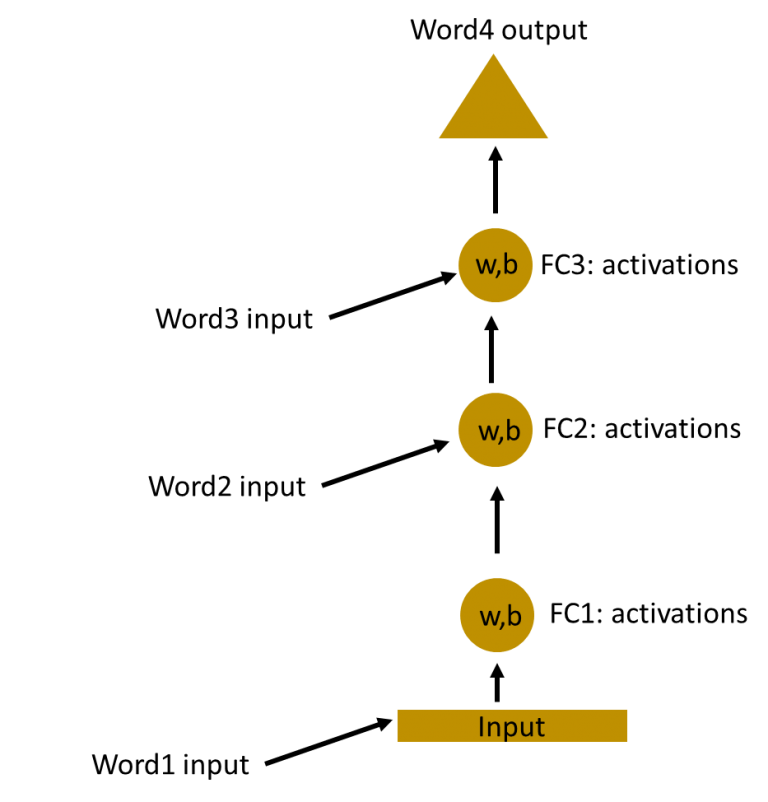


Let’s have a deeper network, where multiple hidden layers are present. So here, the input layer receives the input, the first hidden layer activations are applied and then these activations are sent to the next hidden layer, and successive activations through the layers to produce the output. Each hidden layer is characterized by its own weights and biases.

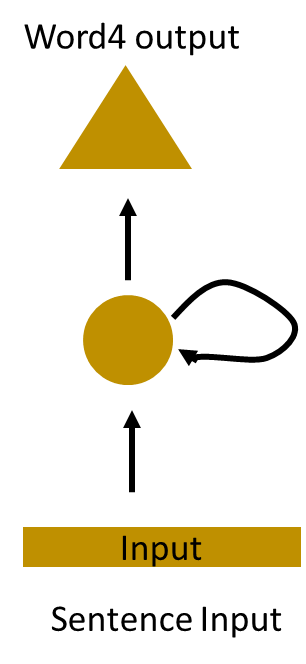
Since each hidden layer has its own weights and activations, they behave independently. Now the objective is to identify the relationship between successive inputs. Can we supply the inputs to hidden layers? Yes we can



Here, the weights and bias of these hidden layers are different. And hence each of these layers behave independently and cannot be combined together. To combine these hidden layers together, we shall have the same weights and bias for these hidden layers.



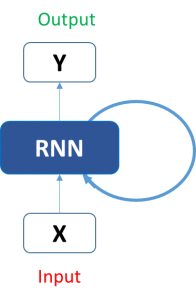
We can now combines these layers together, that the weights and bias of all the hidden layers is the same. All these hidden layers can be rolled in together in a single recurrent layer.



So it’s like supplying the input to the hidden layer. At all the time steps weights of the recurrent neuron would be the same since its a single neuron now. So a recurrent neuron stores the state of a previous input and combines with the current input thereby preserving some relationship of the current input with the previous input.

Understanding a Recurrent Neuron in Detail

Let’s take a simple task at 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 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 be used to predict the fifth letter in the word “hello”. In the above structure, the blue RNN block, applies something called as 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 was “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 –

C:\Users\Admin\Desktop\hidden-state.png

ere, 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 as 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 –

C:\Users\Admin\Desktop\eq2.png

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-

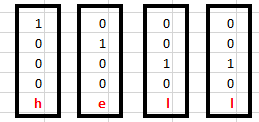
C:\Users\Admin\Desktop\outeq.png  
Let me summarize the steps in a recurrent neuron for you-

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 backpropagated 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 of how we can calculate these states in Excel and get the output.

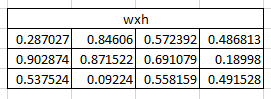
Forward Propagation in a Recurrent Neuron in Excel

Let’s take a look at the inputs first –



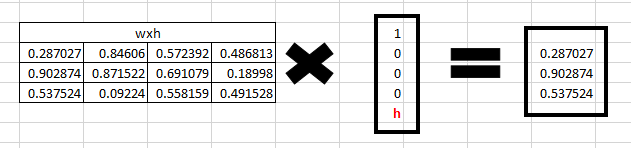
The inputs are one hot encoded. Our entire vocabulary is {h,e,l,o} and hence we can easily one hot encode the inputs.

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 –



Step 1:

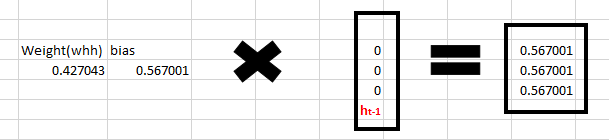
Now for the letter “h”, for the the hidden state we would need Wxh\*Xt. By matrix multiplication, we get it as –



**Step 2:**

**Now moving to the recurrent neuron, we have Whh as the weight which is a 1\*1 matrix as 0.427043  and the bias which is also a 1\*1 matrix as 0.56700 For the letter “h”, the previous state is [0,0,0] since there is no letter prior to it.**

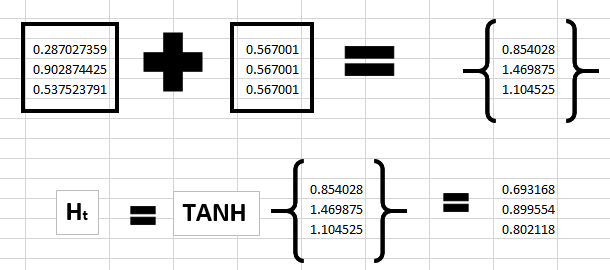
**So to calculate ->  (whh\*ht-1+bias**



Step 3:

Now we can get the current state as –

C:\Users\Admin\Desktop\eq21.png

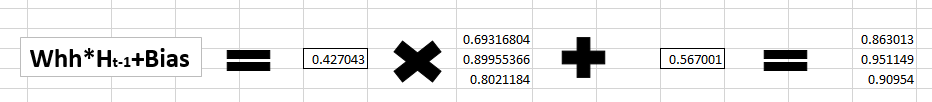
ince for h, there is no previous hidden state we apply the tanh function to this output and get the current state –

Step 4:

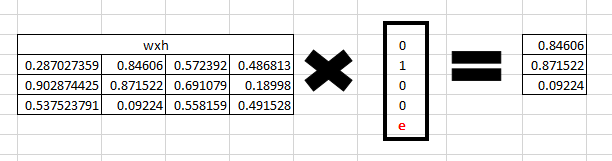
Now we go on to the next state. “e” is now supplied to the network. The processed output of ht, now becomes ht-1, while the one hot encoded e, is xt. Let’s now calculate the current state ht.

C:\Users\Admin\Desktop\eq2 (1).png

Whh\*ht-1 +bias will be –

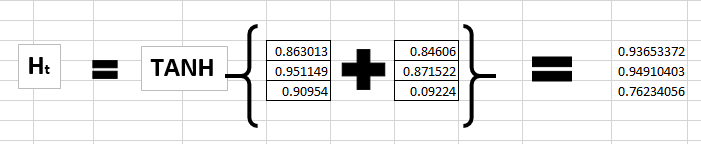


Wxh\*xt will be –



Step 5:

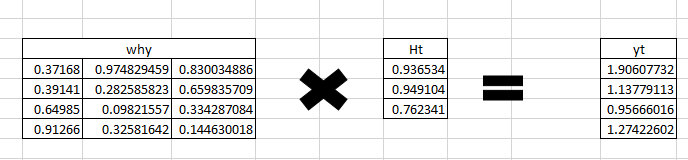
Now calculating ht for the letter “e”,



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.

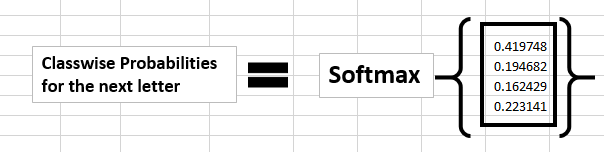
Step 6:

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

C:\Users\Admin\Desktop\outeq (1).png

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)



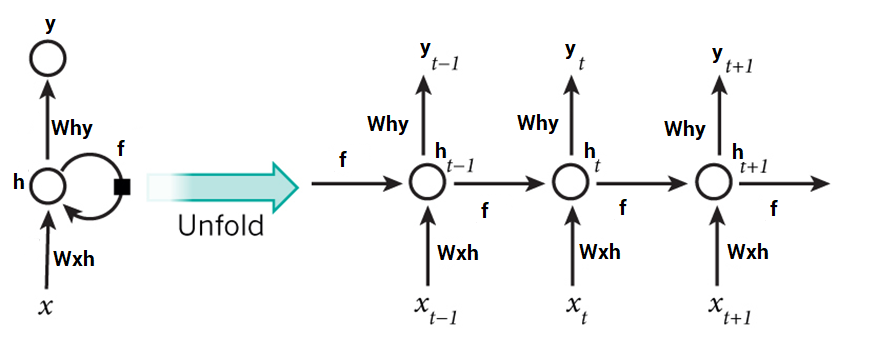
If we convert these probabilities to understand the prediction, we see that the model says that the letter after “e” should be h, since the highest probability is for the letter “h”. Does this mean we have done something wrong? No, so here we have hardly trained the network. We have just shown it two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

Now the next BIG question that faces us is how does Back propagation 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 Back propagation 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. However, I shall be coming up with a detailed article on Recurrent Neural networks with scratch with would have the detailed mathematics of the backpropagation algorithm in a recurrent neural network.