**Recurrent Neural Network**

The sequence of words define their meaning, a time series data – where time defines the occurrence of events, the data of a genome sequence- where every sequence has a different meaning

Sequence of information determines the event itself.

 If we are trying to use such data for any reasonable output, we need a network which has access to some prior knowledge about the data to completely understand it.

**Application**

**Sentiment Classification**

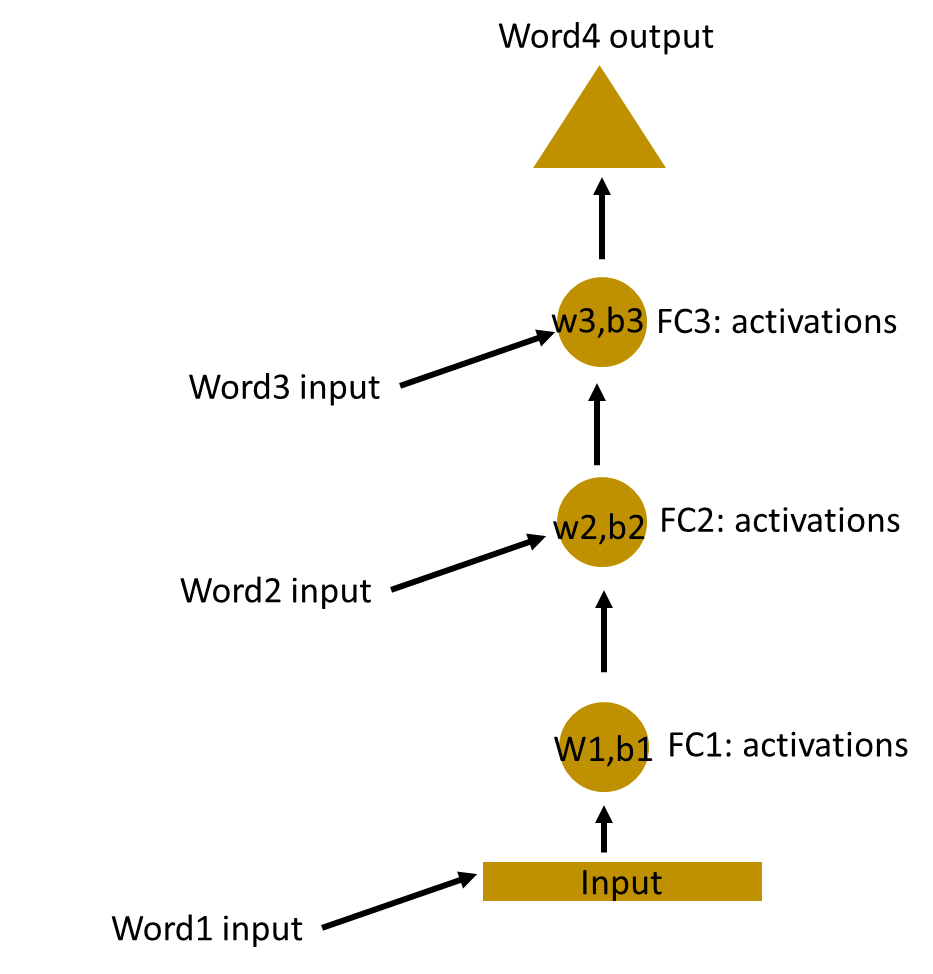
**Image Captioning**

**Language Translation**

**Normal Network**

Weights and bias of hidden layers are different.

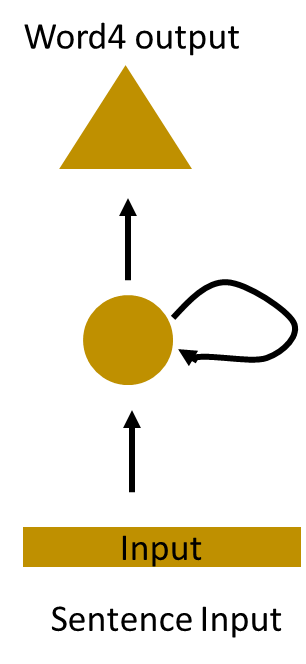
Hidden layers behave independently and cannot be combined together.



**RNN Network**

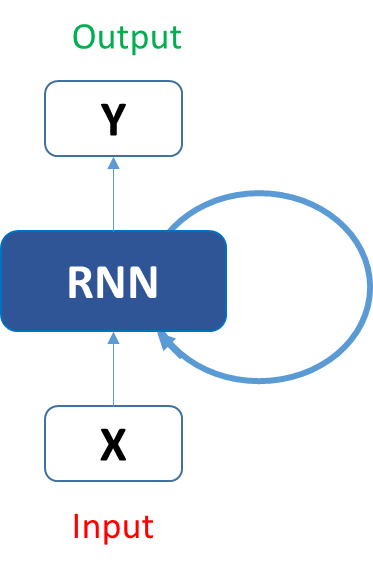
To combine, we shall have the same weights and bias.

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.



HELLO

[H, E, L, L] Predict - O



RNN block, applies something called as **a recurrence formula** to the **input vector** and also its **previous state.**

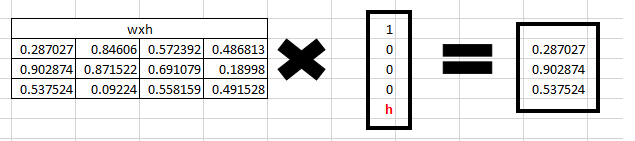
Various time steps of the input



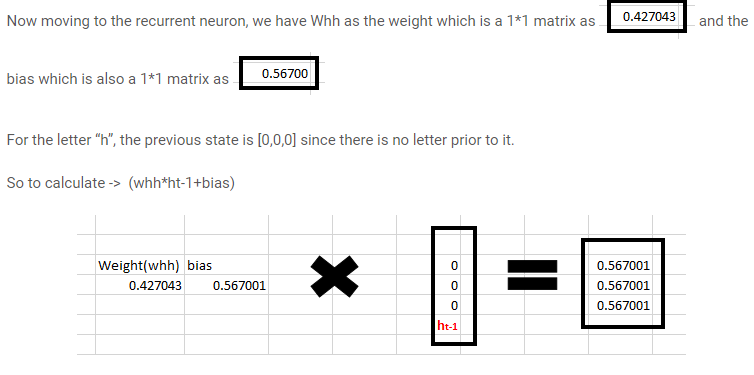


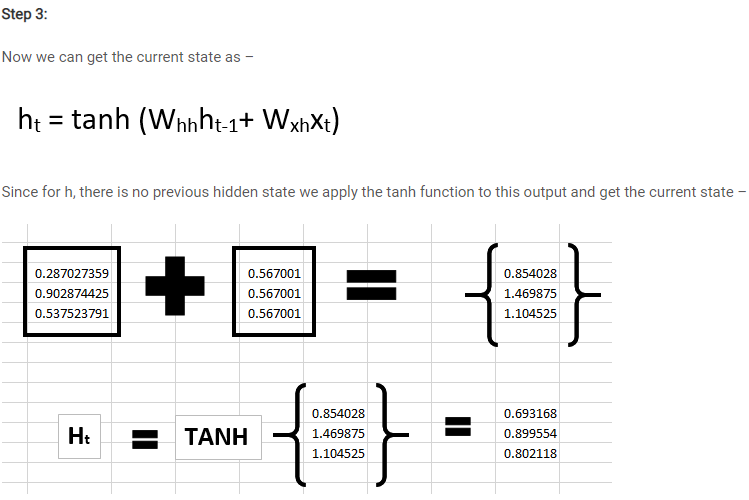


Step-1



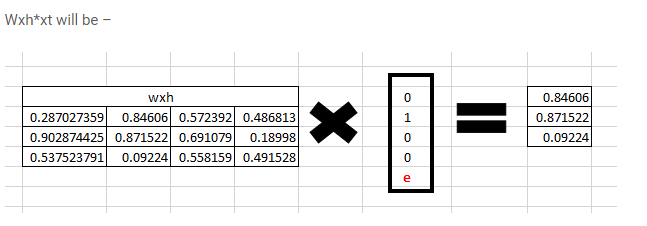
Step 2:

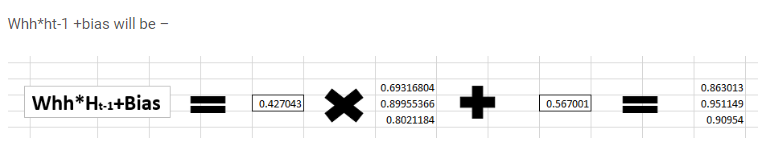


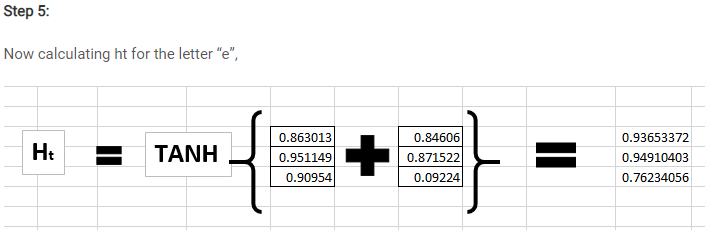


Step 4:

Input e character



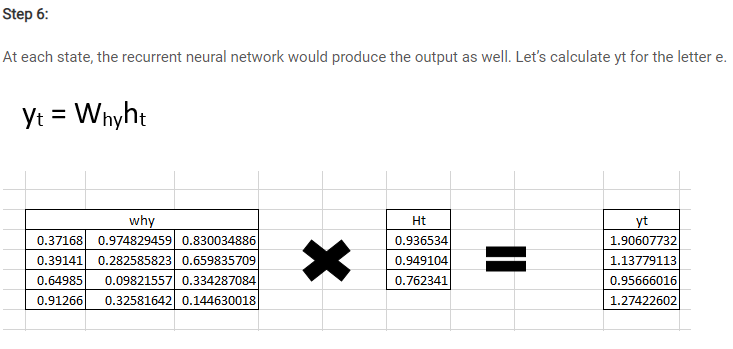


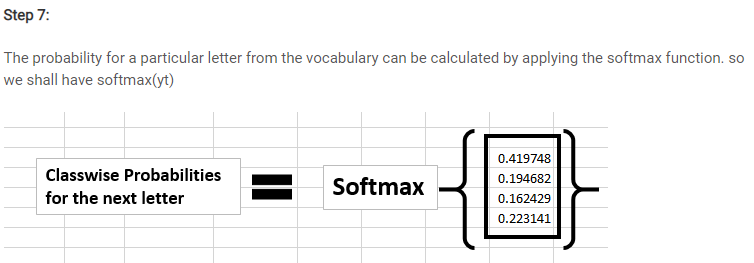


Previous state = tan (Input H + Previous State 0)

Current State = tan (Input e, Previous State H)

Current State\_L = tan (Input L, Previous State e)





**Back propagation in a Recurrent Neural Network (BPTT)**

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).

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 back propagation.

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.

**Vanishing and Exploding Gradient Problem**

RNNs work upon the fact that the result of information is dependent on its previous state or previous n time steps.

Regular RNNs might have a **difficulty in learning** **long range dependencies.**

“The man who ate my pizza has purple hair”

In this case, the description **purple hair** is for the man and **not the pizza**. So this is a long dependency.

If we back propagate the error in this case, we would need to apply the chain rule. To calculate the error after the third time step with respect to the first one –

∂E/∂W = ∂E/∂y3 \*∂y3/∂h3 \*∂h3/∂y2 \*∂y2/∂h1 and there is a long dependency.

Here we apply the chain rule and if any one of the **gradients approached 0,** all the gradients would rush to **zero exponentially** fast due to the **multiplication**.

Such states would no longer help the network to **learn anything**. This is known as the **vanishing gradient problem**.

Vanishing gradient problem is far more threatening as compared to the exploding gradient problem, where the gradients become very large due to a single or multiple gradient values becoming very high.

The reason why Vanishing gradient problem is more concerning is that an exploding gradient problem can be easily solved by clipping the gradients at a predefined threshold value.

Fortunately there are ways to handle vanishing gradient problem as well.

There are architectures like the LSTM (Long Short term memory) and the GRU (Gated Recurrent Units) which can be used to deal with the vanishing gradient problem.

**Other RNN architectures**

RNNs - suffer from vanishing gradient problems - To handle long term dependencies

Severely difficult to train as the number of parameters become extremely large

If we unroll - it becomes so huge - convergence is a challenge

**Long Short Term Memory networks** –special kind of RNN - capable of learning long-term dependencies

LSTMs also have this chain like structure, but the repeating module has a slightly different structure.

Instead of having a single neural network layer, there are multiple layers, interacting in a very special way.

They have an **input gate, a forget gate and an output gate**.

**Gated Recurrent Units** -They are a variant of LSTMs but are simpler in their structure and are easier to train.

Their success is primarily due to the gating network signals that control how the present input and previous memory are used, to update the current activation and produce the current state.

These gates have their own sets of weights that are adaptively updated in the learning phase.

We have just two gates here, the **reset** and the **update** gate

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

<https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/>

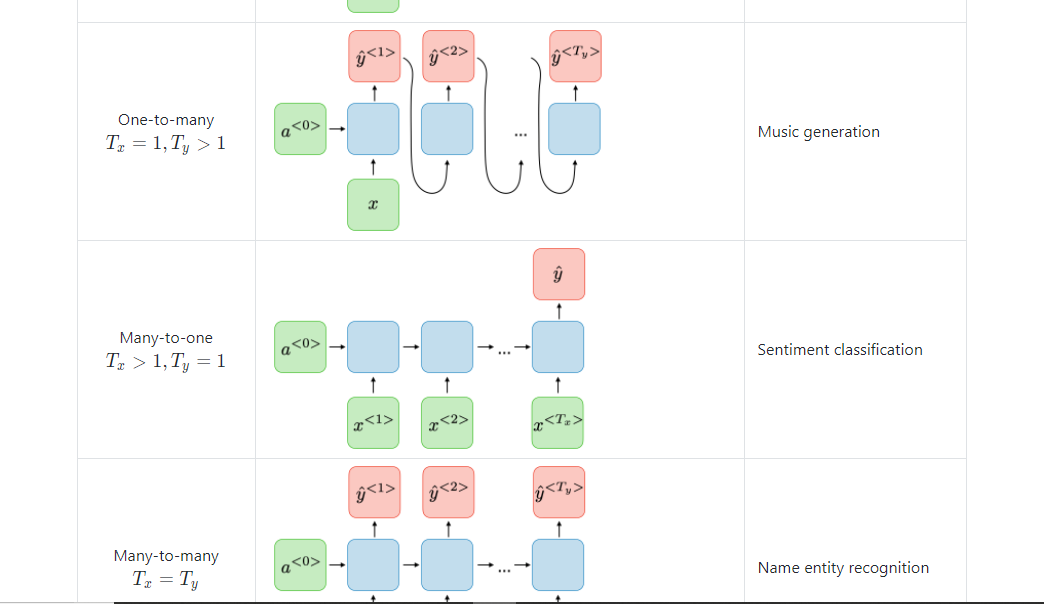
<https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/>

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

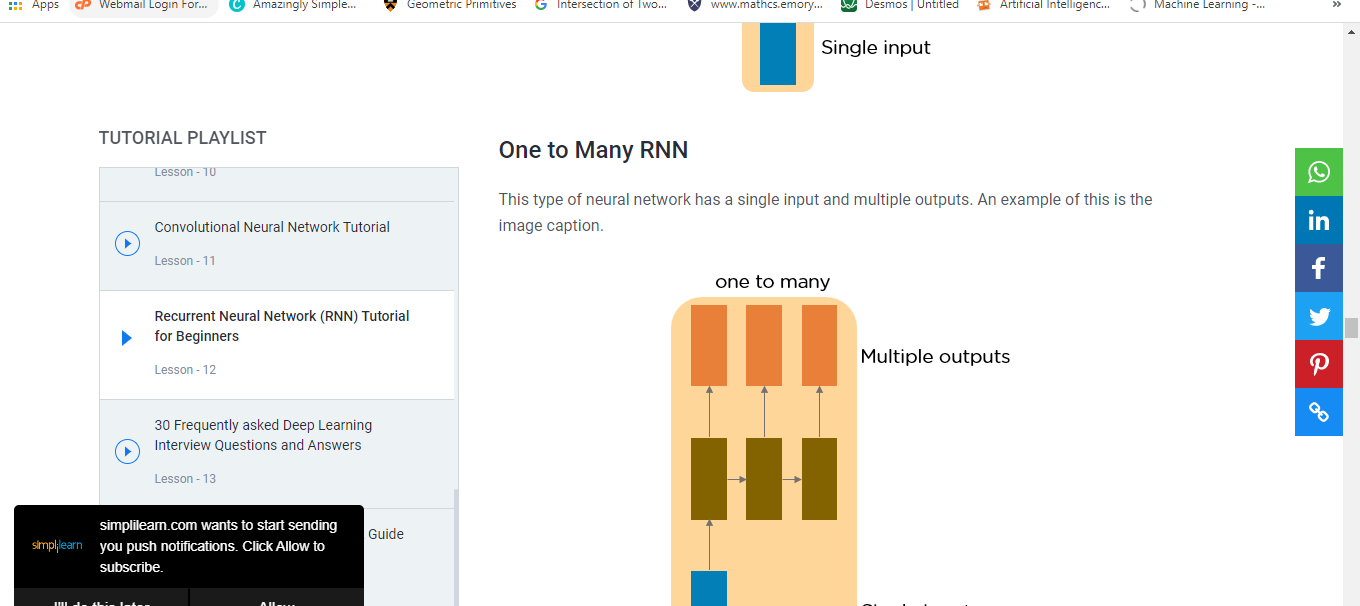
<https://deepai.org/definitions>

<https://www.analyticssteps.com/blogs/learning-recurrent-neural-network-applications-and-its-role-sentiment-analysis>

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>



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



<https://stackoverflow.com/questions/52138290/how-can-we-define-one-to-one-one-to-many-many-to-one-and-many-to-many-lstm-ne>