Lecture-09

Sumit Kumar(15mi446),

Ashish Verma(15mi423)

Anand Vibhuti (15mi401)

<u>Topics-Discuss and explain RNN cell</u>

What is an RNN cell ?

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feed forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input.

Sometimes the context is the single most important thing for the model to predict the most appropriate output.

Let's understand this by an analogy. Suppose you are watching a movie, you keep watching the movie as at any point in time, you have the context because you have seen the movie until that point, then only you are able to relate everything correctly. It means that you remember everything that you have watched.

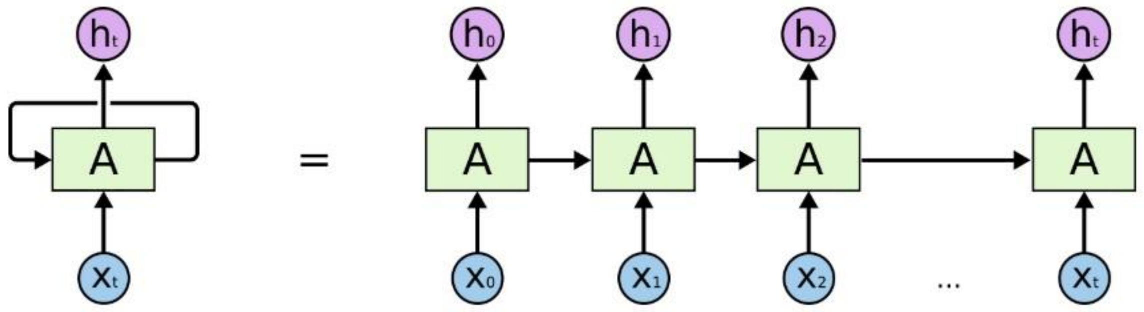
Similarly, RNN remembers everything. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other. Let's say you have to predict the next word in a given sentence, in that case, the relation among all the previous words helps in predicting the better output. The RNN remembers all these relations while training itself.

According to Tensor flow documentation, "An RNN cell, in the most abstract setting, is anything that has a state and performs some operation that takes a matrix of inputs."

RNN cells distinguish themselves from the regular neurons in the sense that they have a state and thus can remember information from the past. RNN cells form the backbone of recurrent networks.

On a mathematical level, a sequence of input are passed through an RNN cell, one at a time. The state of the cell helps it remember the past sequence and combine that information with the current input to provide an output. An easier way to look at it is by unrolling what happens during the sequence which reveals a simpler Deep Network

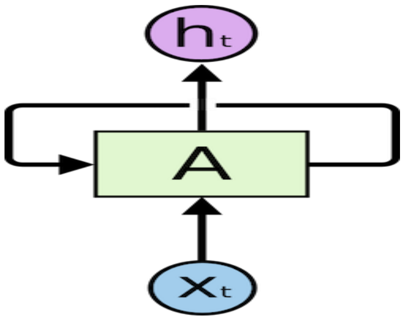
Common RNN cell architectures



The two most commonly used RNN cells are GRUs and LSTMs. Both of these cells have ‘gates’ present in them, which are essentially values between 0 and 1 corresponding to each input. The intuition behind these gates is to forget and retain few selected inputs, showing that these cells can both remember information from the past and also let it go when required. This allows them to deal with sequences better.

recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. ... Unlike feed forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs

In order to achieve it, the RNN creates the networks with loops in them, which allows it to persist the information.



This loop structure allows the neural network to take the sequence of input. If you see the unrolled version, you will understand it better.

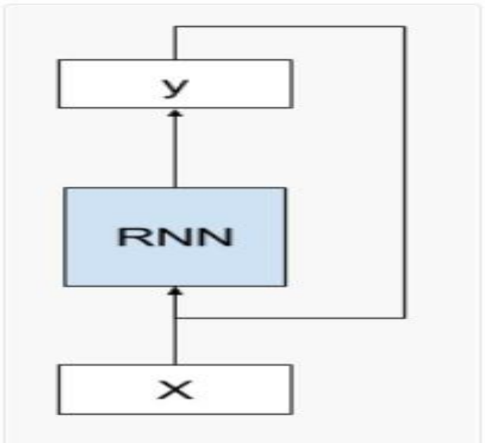
Now a day, RNN has become very popular as it helps in solving many real-life problems which the industries are facing.

Unrolling Recurrent Neural Networks

Recurrent neural networks are a type of neural network where outputs from previous time steps are taken as inputs for the current time step.

We can demonstrate this with a picture.

Below we can see that the network takes both the output of the network from the previous time step as input and uses the internal state from the previous time step as a starting point for the current time step.



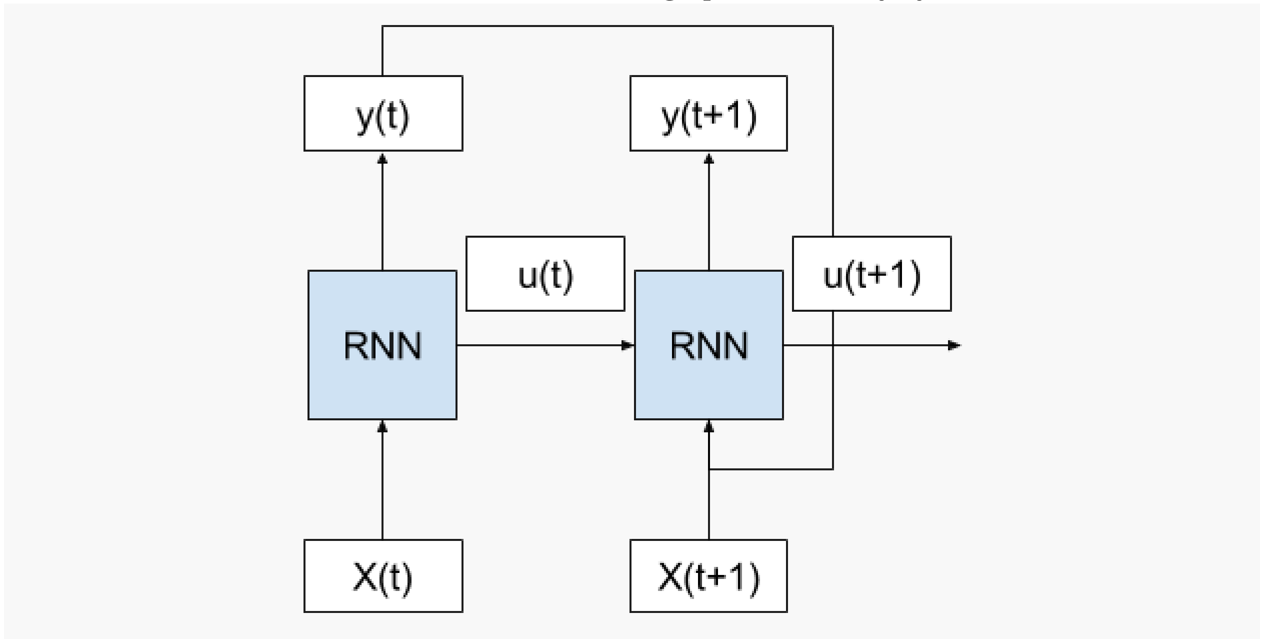
Example of an RNN with a cycle

RNNs are fit and make predictions over many time steps. We can simplify the model by unfolding or unrolling the RNN graph over the input sequence.

Unrolling the Forward Pass

Consider the case where we have multiple time steps of input (X(t), X(t+1), …), multiple time steps of internal state (u(t), u(t+1), …), and multiple time steps of outputs (y(t), y(t+1), …).

We can unfold the above network schematic into a graph without any cycles.



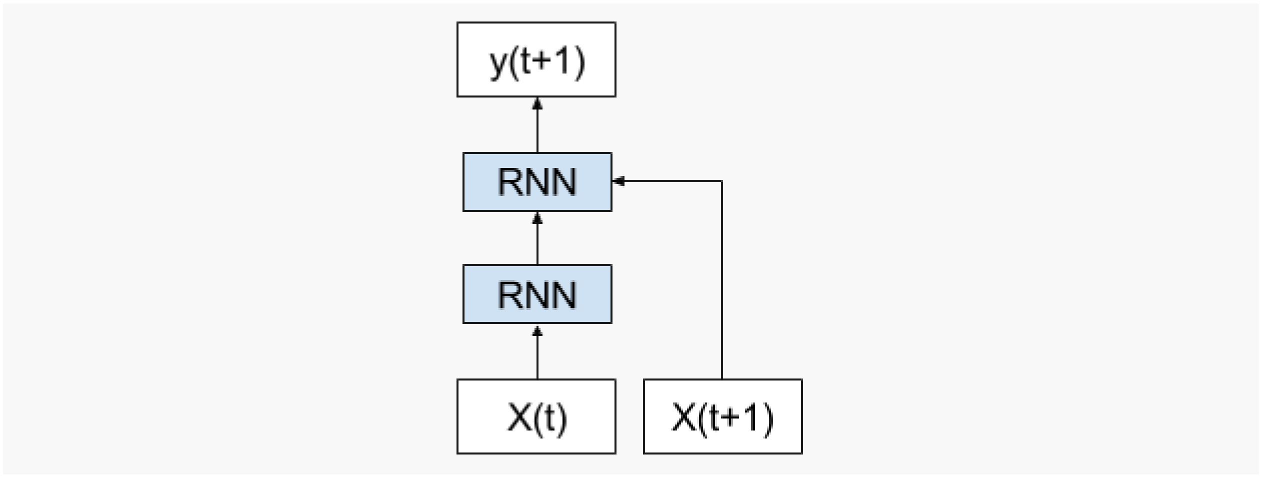
Example of Unrolled RNN on the forward pass

We can see that the cycle is removed and that the output (y(t)) and internal state (u(t)) from the previous time step are passed on to the network as inputs for processing the next time step.

Key in this conceptualization is that the network (RNN) does not change between the unfolded time steps. Specifically, the same weights are used for each time step and it is only the outputs and the internal states that differ.

In this way, it is as though the whole network (topology and weights) are copied for each time step in the input sequence.

Further, each copy of the network may be thought of as an additional layer of the same feed forward neural network.



Example of Unrolled RNN with each copy of the network as a layer

This is a useful conceptual tool and visualization to help in understanding what is going on in the network during the forward pass. It may or may not also be the way that the network is implemented by the deep learning library.

Unrolling the Backward Pass

The idea of network unfolding plays a bigger part in the way recurrent neural networks are implemented for the backward pass.

Unfolding the recurrent network graph also introduces additional concerns. Each time step requires a new copy of the network, which in turn takes up memory, especially for larger networks with thousands or millions of weights. The memory requirements of training large recurrent networks can quickly balloon as the number of time steps climbs into the hundreds.

Back Propagation Through

Time In a recurrent neural network, errors can be propagated further, i.e. more than 2 layers, in order to capture longer history information. This process is usually called unfolding. An unfolded RNN is shown in Figure.

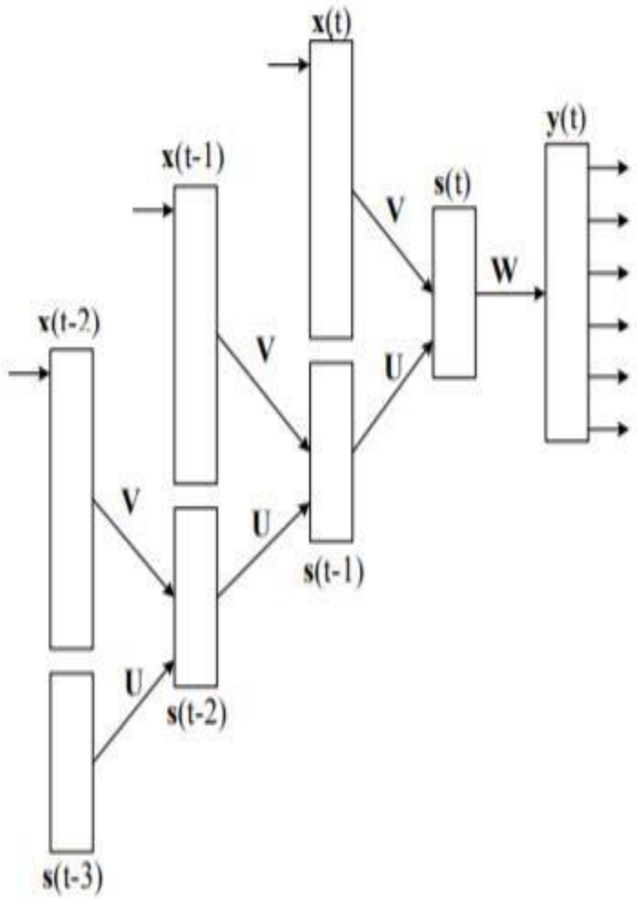
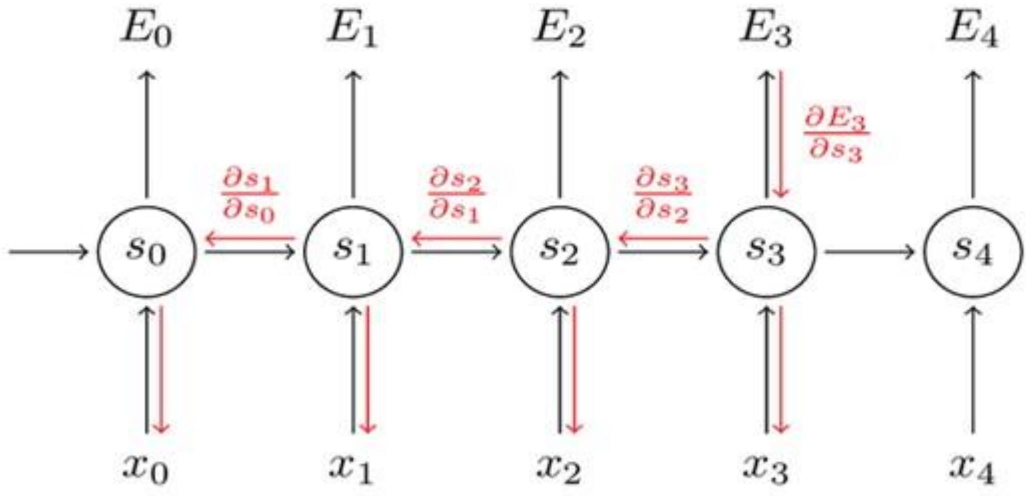


Figure: An unfolded recurrent neural network.


E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

Backpropagation Through Time

Back propagation Through Time (BPTT) is the algorithm that is used to update the weights in the recurrent neural network. One of the common examples of a recurrent neural network is LSTM. Back propagation is an essential skill that you should know if you want to effectively frame sequence prediction problems for the recurrent neural network. You should also be aware of the effects of the Back propagation Through time on the stability, the speed of the system while training the system.

The ultimate goal of the Back propagation algorithm is to minimize the error of the network outputs.

The general algorithm is

1. First, present the input pattern and propagate it through the network to get the output.
2. Then compare the predicted output to the expected output and calculate the error.
3. Then calculate the derivates of the error with respect to the network weights
4. Try to adjust the weights so that the error is Minimum.

The Back propagation algorithm is suitable for the feed forward neural network on fixed sized input-output pairs.

The Back propagation Through Time is the application of Back propagation training algorithm which is applied to the sequence data like the time series. It is applied to the recurrent neural network. The recurrent neural network is shown one input each time step and predicts the corresponding output. So, we can say that BTPP works by unrolling all input time steps. Each time step has one input time step, one output time step and one copy of the network. Then the errors are calculated and accumulated for each time step. The network is then rolled back to update the weights.

But one of the disadvantages of BPTT is when the number of time steps increases the computation also increases. This will make the overall model noisy. The high cost of single parameter updates makes the BPTT impossible to use for a large number of iterations.

This is where Truncated Back propagation comes save the day for us. Truncated Back propagation (TBPTT) is nothing but a slightly modified version of BPTT algorithm for the recurrent neural network. In this, the sequence is processed one time step at a time and periodically the BPTT update is performed for a fixed number of time steps.

The basic Truncated Back propagation algorithm is

1. First, give the sequence of, say K1 time steps of input and output pairs to the network.
2. Then calculate and accumulate the errors across say, k2 time steps by unrolling the network
3. Finally, update the weights by rolling up the network

As you can clearly see that you need two parameters namely k1 and k2 for implementing TBPTT. K1 is the number of forwarding pass time steps between updates. This influences how fast or slow will be the training and the frequency of the weight updates. On the other hand, k2 is the number of time steps which apply to BPTT. It should be large enough to capture the temporal structure in the problem for the network to learn.

Multi-Dimensional Recurrent Neural Networks

Recurrent neural networks (RNNs) have proved effective at one dimensional sequence learning tasks, such as speech and online handwriting recognition. Some of the properties that make RNNs suitable for such tasks, for example robustness to input warping, and the ability to access contextual information, are also desirable in multidimensional domains. However, there has so far been no direct way of applying RNNs to data with more than one spatiotemporal dimension. This paper introduces multi-dimensional recurrent neural networks (MDRNNs), thereby extending the potential applicability of RNNs to vision, video processing, medical imaging and many other areas, while avoiding the scaling problems that have plagued other multidimensional models. Experimental results are provided for two image segmentation tasks.

The following are the few applications of the RNN

- Next word prediction.
- Music composition.
- Image captioning
- Speech recognition
- Time series anomaly detection