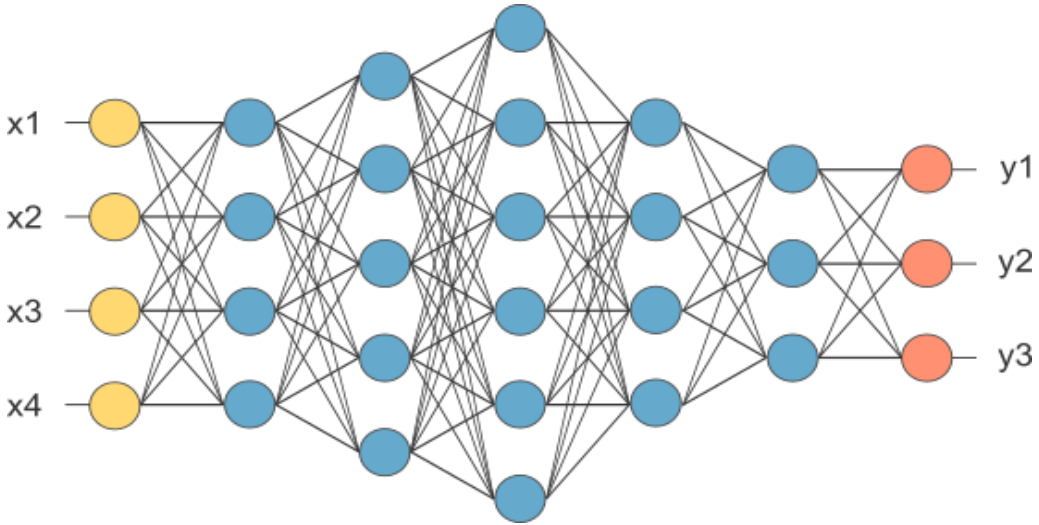
Recurrent Neural Network

Suppose we have a sentence like: – “Artificial intelligence is a very interesting domain”, But instead of this if we say “ **Intelligence is Artificial a very domain interesting**”. Does it make any sense to you? Not really – A small sequence difference or jumble in the words made the sentence incoherent. Understanding this incoherent sentence is tough for human brains so how can we expect a neural network to make sense out of it?

So likely we have multiple other such tasks in everyday life which get completely disrupted or effected when their sequence is disturbed. For example: Working with any particular language – the sequence of words defines and elaborate their own meaning, or you can take the example of time series data – where time is the main key and defines the occurrence of events. Then we need to maintain the sequence because where every sequence has a different meaning and importance.

This is why recurrent neural networks come into the picture which can maintain the sequence of the input data throughout the process.

RNN



This is our fully connected network. If x_1 ... x_n, n is very large and growing, this network would become too large. We now will input one x_i at a time, and re-use the same edge weights.

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.

Why Recurrent Neural Networks?

RNN were created because there were a few issues in the feed-forward neural network:

Cannot handle sequential data

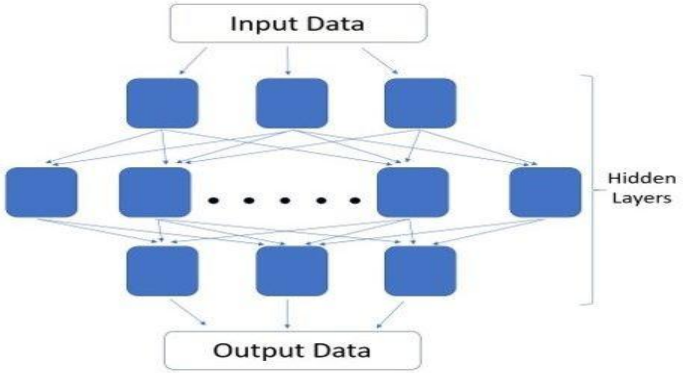
Considers only the current input

Cannot memorize previous inputs

The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

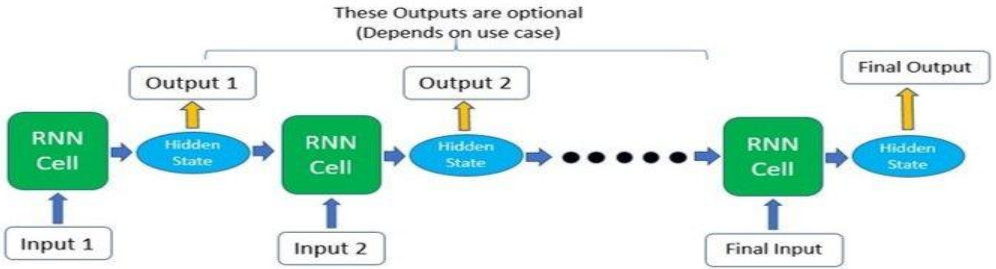
Comparison Between:

Traditional Feed-Forward Network



VS

Recurrent Neural Network



Traditional feed-forward neural networks take in a fixed amount of input data all at the same time and produce a fixed amount of output each time.

On the other hand, RNNs do not consume all the input data at once. Instead, they take them in one at a time and in a sequence.

At each step, the RNN does a series of calculations before producing an output.

The output, known as the hidden state, is then combined with the next input in the sequence to produce another output. This process continues until the model is programmed to finish or the input sequence ends.

Getting targets when modeling sequences

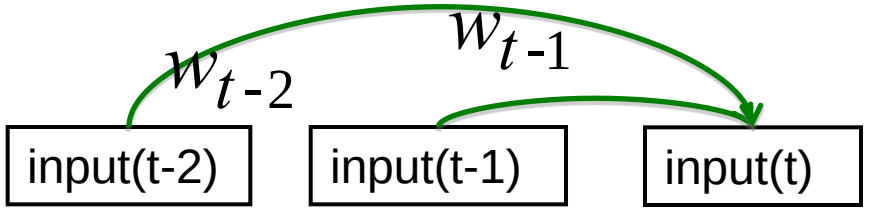
- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.

- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
  - The target output sequence is the input sequence with an advance of 1 step.
  - This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
  - For temporal sequences there is a natural order for the predictions.

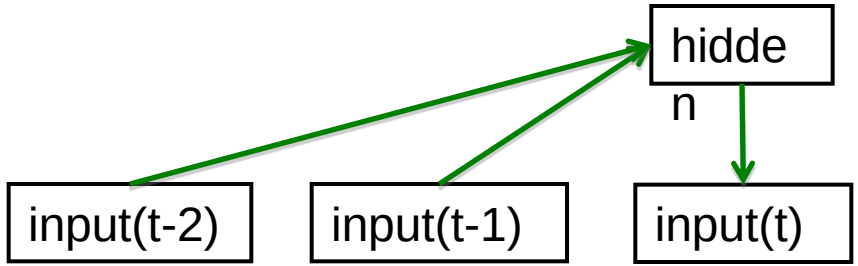
- Predicting the next term in a sequence blurs the distinction between supervised and unsupervised learning.
  - It uses methods designed for supervised learning, but it doesn’t require a separate teaching signal.

Memoryless models for sequences

- Autoregressive models
  Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.



- Feed-forward neural nets
  These generalize autoregressive models by using one or more layers of non-linear hidden units.

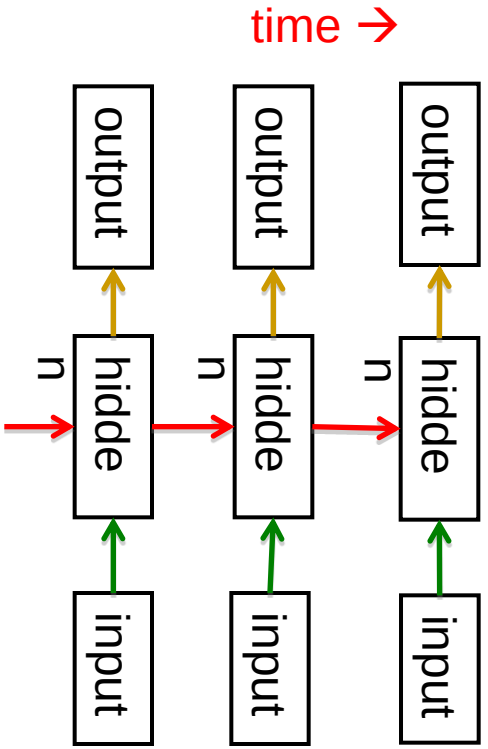


Beyond memoryless models

- If we give our generative model some hidden state, and if we give this hidden state its own internal dynamics, we get a much more interesting kind of model.
  - It can store information in its hidden state for a long time.
  - If the dynamics is noisy and the way it generates outputs from its hidden state is noisy, we can never know its exact hidden state.
  - The best we can do is to infer a probability distribution over the space of hidden state vectors.
- This inference is only tractable for two types of hidden state model.

Recurrent neural networks

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.



Recurrent neural networks

- What kinds of behaviour can RNNs exhibit?
  - They can oscillate. Good for motor control?
  - They can settle to point attractors. Good for retrieving memories?
  - They can behave chaotically. Bad for information processing?
  - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects.
- But the computational power of RNNs makes them very hard to train.
  - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer).

Applications:

1. Language Modelling and Generating Text

Taking a sequence of words as input, we try to predict the possibility of the next word. This can be considered to be one of the most useful approaches for translation since the most likely sentence would be the one that is correct. In this method, the probability of the output of a particular time-step is used to sample the words in the next iteration.

2. Machine Translation

RNNs in one form or the other can be used for translating text from one language to other . Almost all of the Translation systems being used today use some advanced version of a RNN. The input can be the source language and the output will be in the target language which the user wants.

3. Speech Recognition

RNNs can be used for predicting phonetic segments considering sound waves from a medium as an input source .The set of inputs consists of phoneme or acoustic signals from an audio which are processed in a proper manner and taken as inputs. The RNN network will compute the phonemes and then produce a phonetic segment along with the likelihood of output.

4. Generating Image Descriptions

A combination of CNNs and RNNs are used to provide a description of what exactly is happening inside an image. CNN does the segmentation part and RNN then uses the segmented data to recreate the description.

5. Video Tagging

RNNs can be used for video search where we can do image description of a video divided into numerous frames.

6. Text Summarization

This application can provide major help in summarizing content from literatures and customising them for delivery within applications which cannot support large volumes of text. For example, if a publisher wants to display the summary of one of his books on its backpage to help the readers get an idea of the content present within, Text Summarization would be helpful.

7. Call Center Analysis

This can be considered as one of the major applications of RNNs in the field of audio processing.

8. Face detection, OCR Applications as Image Recognition

Image recognition is one of the major applications of computer vision. It is also one of the most accessible form of RNN to explain.

Long Short-Term Memory Networks (LSTM):

Sequence prediction problems have been around for a long time. They are considered as one of the hardest problems to solve in the data science industry. These include a wide range of problems; from predicting sales to finding patterns in stock markets' data, from understanding movie plots to recognizing your way of speech, from language translations to predicting your next word on your iPhone's keyboard.
With the recent breakthroughs that have been happening in data science, it is found that for almost all of these sequence prediction problems, Long short Term Memory networks, a.k.a LSTMs have been observed as the most effective solution.

LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time. The purpose of this article is to explain LSTM and enable you to use it in real life problems

Limitations of RNNs

Recurrent Neural Networks work just fine when we are dealing with short-term dependencies. That is when applied to problems like:

The colour of the sky is _____.

RNNs turn out to be quite effective. This is because this problem has nothing to do with the context of the statement. The RNN need not remember what was said before this, or what was its meaning, all they need to know is that in most cases the sky is blue. Thus the prediction would be:

The colour of the sky is blue.

However, vanilla RNNs fail to understand the context behind an input. Something that was said long before, cannot be recalled when making predictions in the present. Let's understand this as an example:

I spent 20 long years working for the under-privileged kids in Spain. I then moved to Africa.

.....

I can speak fluent _____.

RNN remembers things for just small durations of time, i.e. if we need the information after a small time it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the Long Short-Term Memory Networks.

Improvement over RNN: LSTM (Long Short-Term Memory) Networks

When we arrange our calendar for the day, we prioritize our appointments right? If in case we need to make some space for anything important we know which meeting could be canceled to accommodate a possible meeting.

Turns out that an RNN doesn't do so. In order to add a new information, it transforms the existing information completely by applying a function. Because of this, the entire information is modified, on the whole, i. e. there is no consideration for *'important'* information and *'not so important'* information.

LSTMs on the other hand, make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. The information at a particular cell state has three different dependencies.

We'll visualize this with an example. Let's take the example of predicting stock prices for a particular stock. The stock price of today will depend upon:

The trend that the stock has been following in the previous days, maybe a downtrend or an uptrend.

The price of the stock on the previous day, because many traders compare the stock's previous day price before buying it.

The factors that can affect the price of the stock for today. This can be a new company policy that is being criticized widely, or a drop in the company's profit, or maybe an unexpected change in the senior leadership of the company.

Architecture of LSTMs

The functioning of LSTM can be visualized by understanding the functioning of a news channel's team covering a murder story. Now, a news story is built around facts, evidence and statements of many people. Whenever a new event occurs you take either of the three steps.

Let's say, we were assuming that the murder was done by 'poisoning' the victim, but the autopsy report that just came in said that the cause of death was 'an impact on the head'. Being a part of this news team what do you do? You immediately **forget** the previous cause of death and all stories that were woven around this fact.

What, if an entirely new suspect is introduced into the picture. A person who had grudges with the victim and could be the murderer? You **input** this information into your news feed, right?

Now all these broken pieces of information cannot be served on mainstream media. So, after a certain time interval, you need to summarize this information and **output** the relevant things to your audience. Maybe in the form of "*XYZ turns out to be the prime suspect.*".

Now let's get into the details of the architecture of LSTM network: