RNN

Recurrent neural networks (RNN) are a class of neural networks that are helpful in modeling sequence data and are used by Apple's Siri and Google's voice search. Sequential data is basically just ordered data in which related things follow each other. The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what's coming next. This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more.

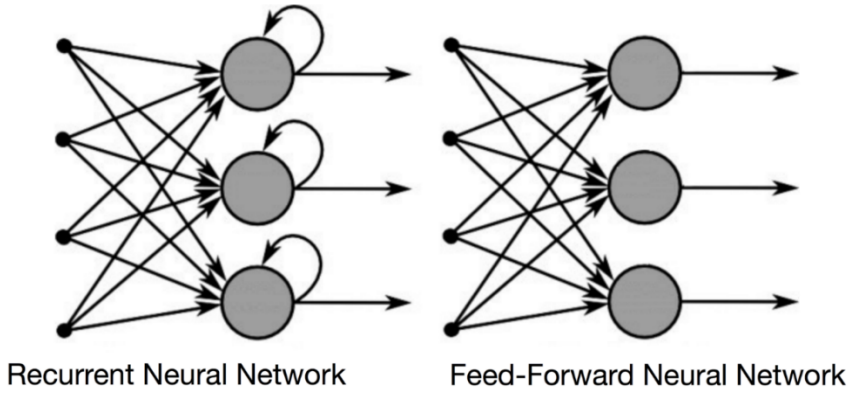
The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Recurrent means the output at the current time step becomes the input to the next time step. At each element of the sequence, the model considers not just the current input, but what it remembers about the preceding elements. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps

There are many instances where data naturally forms sequences and in those cases, order and content are equally important. Other examples of sequence data include video, music, DNA sequences, and many others. When learning from sequence data, short term memory becomes useful for processing a series of related data with ordered context. For this, machine learning researchers have long turned to the recurrent neural network, or RNN.

RNN vs Feed Forward Neural Network (ANN CNN)

- In a feed-forward neural network, the information only moves in one direction i.e. from the input layer through the hidden layers to the output layer. The information moves straight through the network and never touches a node twice.

In a RNN the information cycles through a loop.



- When Feed Forward neural network makes a decision, it considers only the current input. Because a feed-forward network only considers the current input and have no memory of the input they receive and are bad at predicting what's coming next. It simply can't remember anything about what happened in the past except its training.

When RNN makes a decision, it considers the current input and also what it has learned from the inputs it received previously (the output from previous step are fed as input to the current step). Simply putting recurrent neural networks add the immediate past to the present. Therefore, a RNN has two inputs i.e. the present and the recent past. RNN have a memory which remembers all information about what has been calculated. A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory.

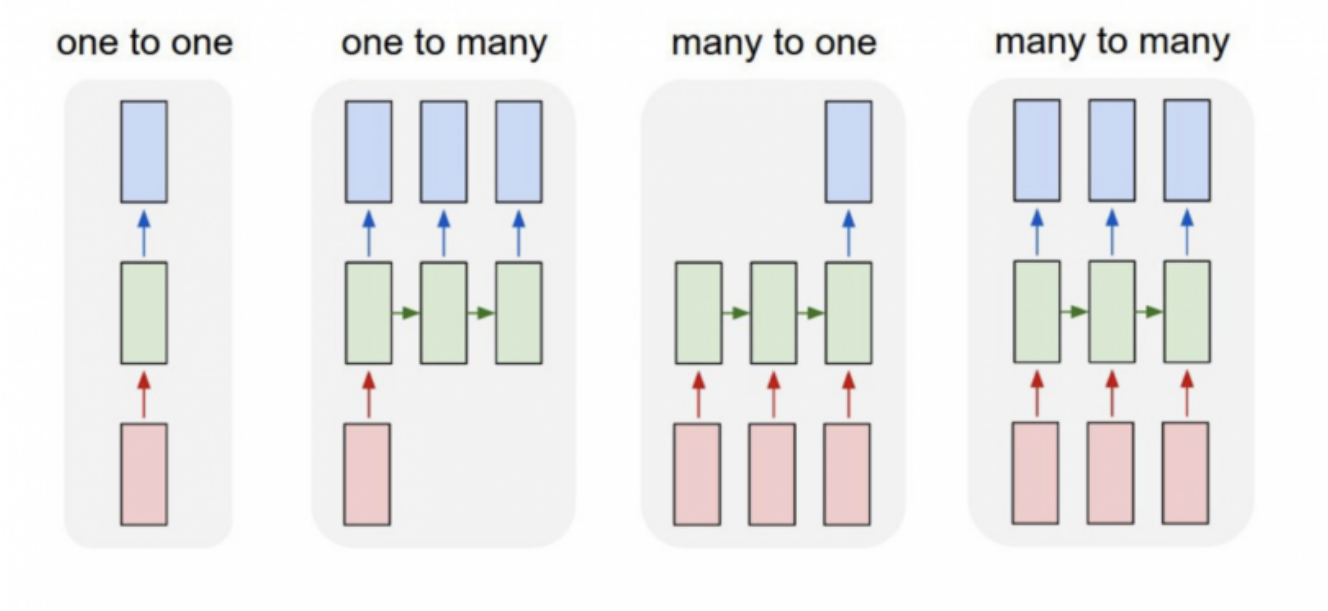
Imagine you have a normal feed-forward neural network and give it the word "neuron" as an input and it processes the word character by character. By the time it reaches the character "r," it has already forgotten about "n," "e" and "u," which makes it almost impossible for this type of neural network to predict which character would come next. A recurrent neural network, however, is able to remember those characters because of its internal memory. It produces output, copies that output and loops it back into the network.

- A feed-forward neural network assigns like all other deep learning algorithms a weight matrix to its inputs and then produces the output.

RNNs apply weights to the current and also to the previous input.

- Feed-forward neural networks map one input to one output

RNNs can map one to many, many to many (translation) and many to one (classifying a voice).



One to One

One to One RNN is the most basic and traditional type of Neural network giving a single output for a single input.

One to Many

One to Many is a kind of RNN which is applied in situations that give multiple output for a single input. A basic example of its application would be Music generation. In Music generation models, RNN models are used to generate a music piece (multiple output) from a single musical note (single input).

Many to One

Many-to-one RNN is usually seen for sentiment analysis model as a common example. As the name suggests, this kind of model is used when multiple inputs are required to give a single output.

For example: The Twitter sentiment analysis model. In that model, a text input (words as multiple inputs) gives its fixed sentiment (single output). Another example could be movie ratings model that takes review texts as input to provide a rating to a movie that may range from 1 to 5.

Many-to-Many

Many-to-Many RNN takes multiple input and gives multiple output but Many-to-Many models can be two kinds-

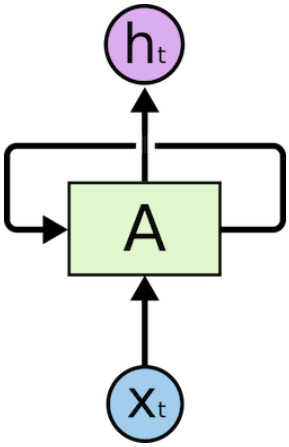
When input and output layers have the same size. This can be also understood as every input having a output, and a common application can be found in Named Entity Recognition.

Where input and output layers are of different size, and the most common application of this kind of RNN architecture is seen in Machine Translation. For example, "I Love you", the 3 magical words of the English language translates to only 2 in Spanish, "te amo".

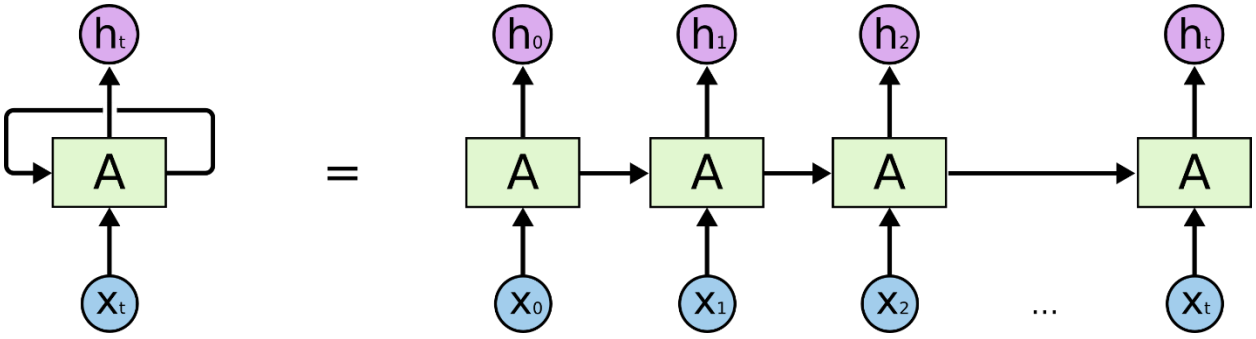
How RNN works

Sometimes the context is the single most important thing for the model to predict the most appropriate output. Let's understand this by another 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. The decision a recurrent net reached at time step t-1 affects the decision it will reach one moment later at time step t. So recurrent networks have two sources of input, the present and the recent past which combine to determine how they respond to new data.

It is often said that recurrent networks have memory. Adding memory to neural networks has a purpose. Recurrent nets use it to perform tasks that feedforward networks can't. That sequential information is preserved in the recurrent network's hidden state which manages to span many time steps as it cascades forward to affect the processing of each new example. It is finding correlations between events separated by many moments, and these correlations are called long-term dependencies because an event downstream in time depends upon, and is a function of, one or more events that came before.



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



As you can see in the unrolled version. First, it takes the x(0) from the sequence of input and then it outputs h(0) which together with x(1) is the input for the next step. So, the h(0) and x(1) is the input for the next step. Similarly h(1) from the next is the input with x(2) for the next step and so on. This way, it keeps remembering the context while training. While processing, it passes the previous hidden state to the next step of the sequence. The hidden state acts as the neural networks memory. It holds information on previous data the network has seen before.

The basic building block of an RNN is a computational unit (or cell) that combines two inputs - the data of the current time step in the sequence and the unit's own output from the previous time step. Formally, an RNN cell is a non-linear transformation that maps the input signal x at time t and the hidden state h of the previous time step to the current hidden state

The hidden state at time step t is ht. It is a function of the input at the same time step xt, modified by a weight matrix W added to the hidden state of the previous time step ht-1 multiplied by its own hidden-state-to-hidden-state matrix U, otherwise known as a transition matrix and similar to a Markov chain. The weight matrices are filters that determine how much importance to accord to both the present input and the past hidden state. First the previous hidden state and input are combined to form a vector. That vector now has information on the current input and previous inputs. The vector goes through the tanh activation and the output is the new hidden state or the memory of the network. The error they generate will return via backpropagation and be used to adjust their weights until error can't go any lower

Formula for calculating current state

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

or

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

where

whh -> weight at recurrent neuron

wxh -> weight at input neuron

Formula for calculating output

$$y_t = W_{hy} h_t$$

Yt -> output

Why -> weight at output layer

Steps

- A single time step of the input is provided to the network.

- Then calculate its current state using set of current input and the previous state.

- The current ht becomes ht-1 for the next time step.

- One can go as many time steps according to the problem and join the information from all the previous states.

- Once all the time steps are completed the final current state is used to calculate the output.

- The output is then compared to the actual output i.e. the target output and the error is generated.

- The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

Problems with RNN

- Long-Term Dependencies problem

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

RNNs are good in handling sequential data but they run into problem when the context is far away. Example: I live France and I know ____. The answer must be 'French' here but if the there are some more words in between 'I live in France' & 'I know ____'. It'll be difficult for RNNs to predict 'French'. This is the problem of Long-Term Dependencies.

- Vanishing Gradients

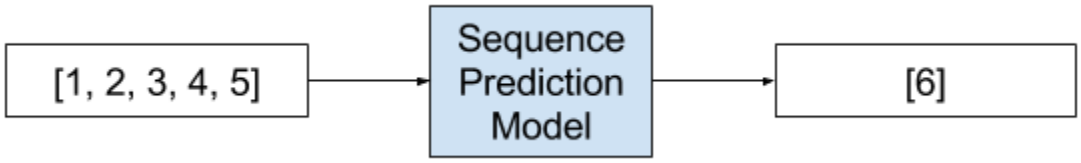
- Exploding Gradients

Applications

Sequence prediction is different to other types of supervised learning problems. The sequence imposes an order on the observations that must be preserved when training models and making predictions. There are four main types of sequence prediction problems:

- Sequence Prediction

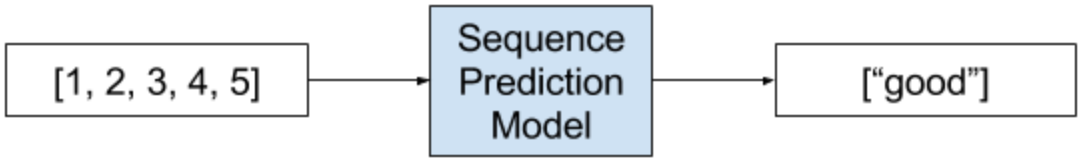
Given an input sequence, predict the next value in the sequence.



Examples of Sequence Prediction Problems are Weather Forecasting, Stock Market Prediction, Product Recommendation

- Sequence Classification

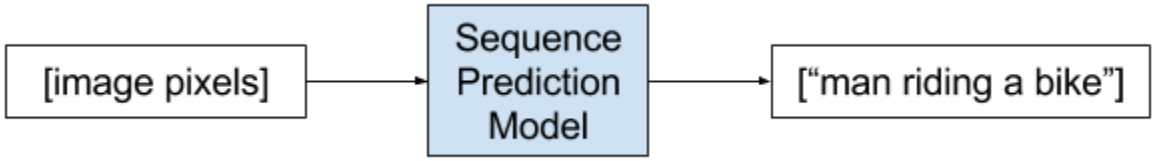
Given an input sequence, classify the sequence.



Examples of Sequence Classification Problems are Sentiment Analysis, DNA Sequence Classification, Anomaly Detection.

- Sequence Generation

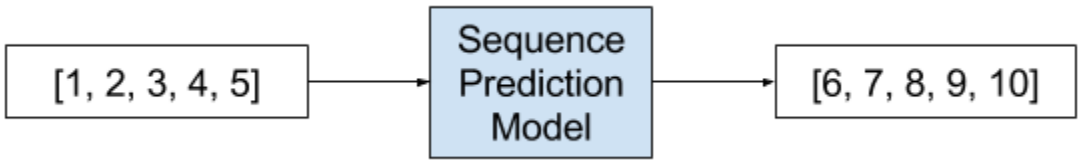
Given an observation, generate an output sequence.



Examples of Sequence Generation Problems are Image Captioning, Text Generation, Music Generation.

- Sequence-to-Sequence Prediction

Given an input sequence, generate an output sequence.

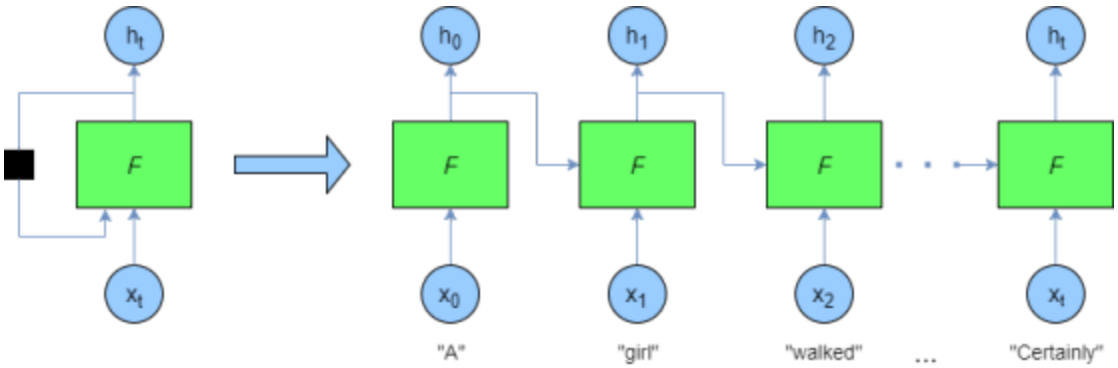


Examples of Sequence-to-Sequence Prediction Problems are Multi-Step Time Series Forecasting, Language Translation, Text Summarization.

Example

Consider the following text string: “A girl walked into a bar, and she said ‘Can I have a drink please?’.

The bartender said ‘Certainly {}”. There are many options for what could fill in the {} symbol in the above string, for instance, “miss”, “ma’am” and so on. However, other words could also fit, such as “sir”, “Mister” etc. In order to get the correct gender of the noun, the neural network needs to “recall” that two previous words designating the likely gender (i.e. “girl” and “she”) were used. This type of flow of information through time (or sequence) in a recurrent neural network is shown in the diagram below, which unrolls the sequence.



This network shows how we can supply a stream of data to the recurrent neural network. For instance, first we supply the word vector for “A” (more about word vectors later) to the network F - the output of the nodes in F are fed into the “next” network and also act as a stand-alone output (h0). The next network (though it is really the same network) F at time t=1 takes the next word vector for “girl” and the previous output h0 into its hidden nodes, producing the next output h1 and so on.

Architectures

Fully Recurrent Neural Network

Recursive Neural Network

Hopfield Network

Elman Networks and Jordan Networks or Simple Recurrent Network(SRN)

Echo State Network

Neural History Compressor

Long Short Term Memory(LSTM)

Gated Recurrent Unit

Bi-Directional Recurrent Neural Network

Continuous-Time Recurrent Neural Network(CTRNN)

Hierarchical Recurrent Neural Network

Recurrent Multilayer Perceptron Network

Multiple Timescales Model

Neural Turing Machines(NTM)

Differentiable Neural Computer(DNC)

Neural Network Pushdown Automata(NNPDA)