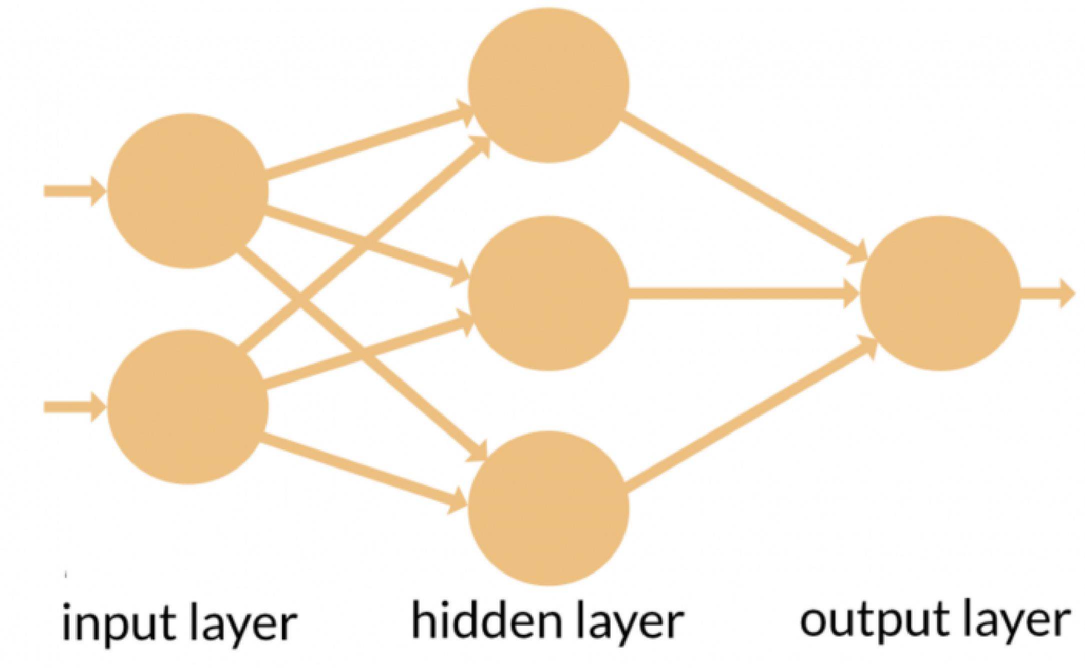
Standard RNN

A basic understanding of "conventional" feed-forward neural networks, as well as sequential data, is required in order to fully appreciate RNNs.

Essentially, sequential data is merely organised data in which related items are listed sequentially. Data such as financial statistics or the DNA sequence are a couple of examples. Time series data, which is just a list of data points in chronological order, is probably the most used sort of sequential data.

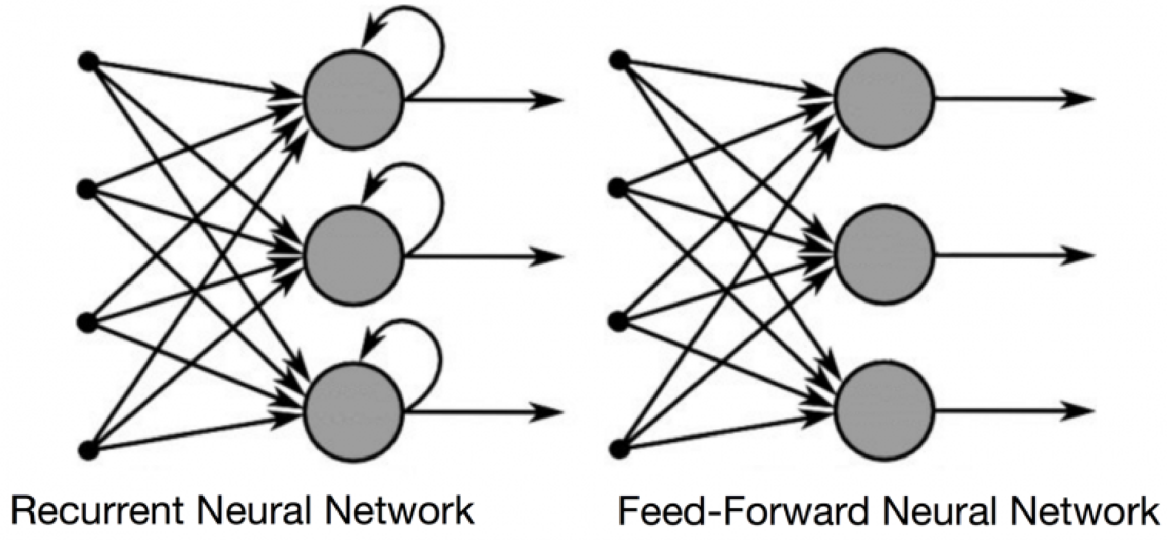
FEED-FORWARD NEURAL NETWORKS VS. RNN



Feed-forward neural networks, also known as recurrent neural networks, get their names from the way they process information. There is only one route in which information can go in a feed-forward neural network: from the input layer to the output layer, via the hidden layers. Because the data travels in a straight line, it never sees another node twice.

No previous input is stored in feed-forward neural networks, which makes them unreliable at predicting what will happen next. A feed-forward network has no concept of time order because it only analyses the current input. It simply has no recollection of the events of the past other than what it was taught.
The data in an RNN is fed into a loop. This decision-making algorithm takes into account both the current input and what it has learned from past inputs.

The following two images demonstrate the differences between an RNN and a feed-forward neural network in terms of information flow.



A short-term memory is present in a typical RNN. They have a long-term memory when used with an LSTM (more on that later).

You can also use an example to help convey the concept of recurrent neural networks' memory:

The word "neuron" is fed into the feed-forward neural network, which analyses it letter by character. There is no way for this form of neural network to predict which character will come next until it has already forgotten the preceding characters, such as "n," "e," and "u.".

However, the internal memory of a recurrent neural network allows it to recall those characters. Once the data has been processed, a copy is made and the same process is repeated.

Recurrent neural networks, in other words, bring the recent past into the present.

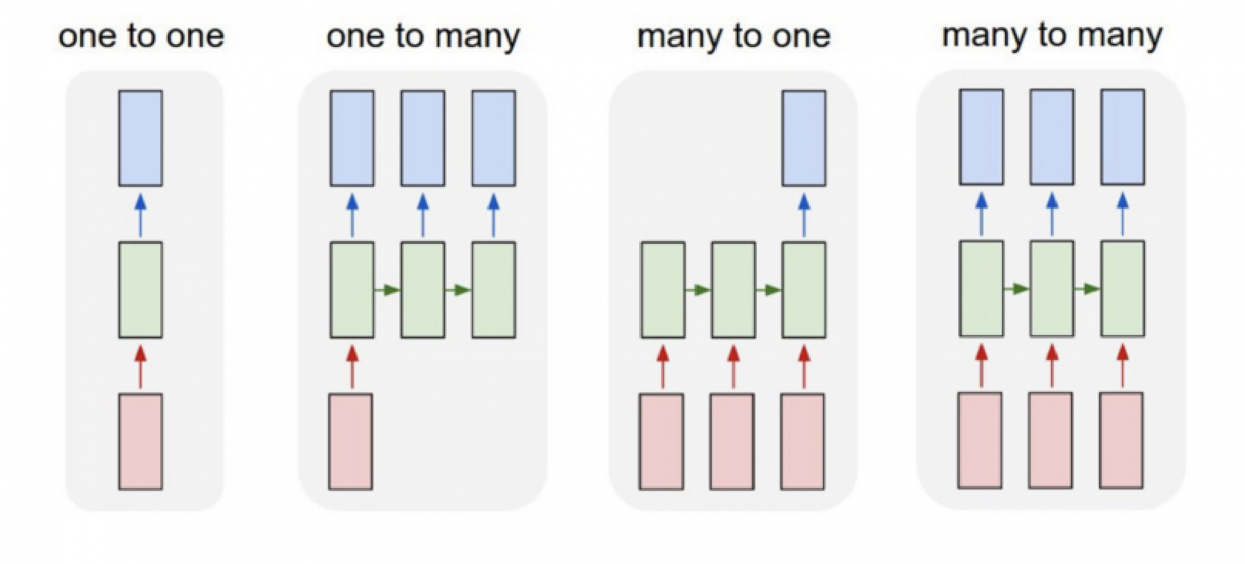
As a result, an RNN uses the present and recent past as inputs. Data sequence contains critical information about what will happen next, which makes it possible for an RNN to perform tasks other algorithms cannot.

When using a feed-forward neural network, a weight matrix is assigned to each of the inputs before the output is generated. It should be noted that RNNs use both the current and previous input as weights in their calculations.There are two additional ways that a recurrent neural network can modify its weights: gradient descent and backpropagation (BPTT).

Recurrent neural network types

- One to One
- One to Many
- Many to One
- Many to Many

Also, keep in mind that RNNs can map one input to many, many to many (translation), and many to one while feed-forward neural networks map one input to one output (classifying a voice).



Backpropagation Through Time

To fully grasp the concept of backpropagation across time, it is necessary to first grasp the concepts of forward and backpropagation.

WHAT IS BACKPROPAGATION?

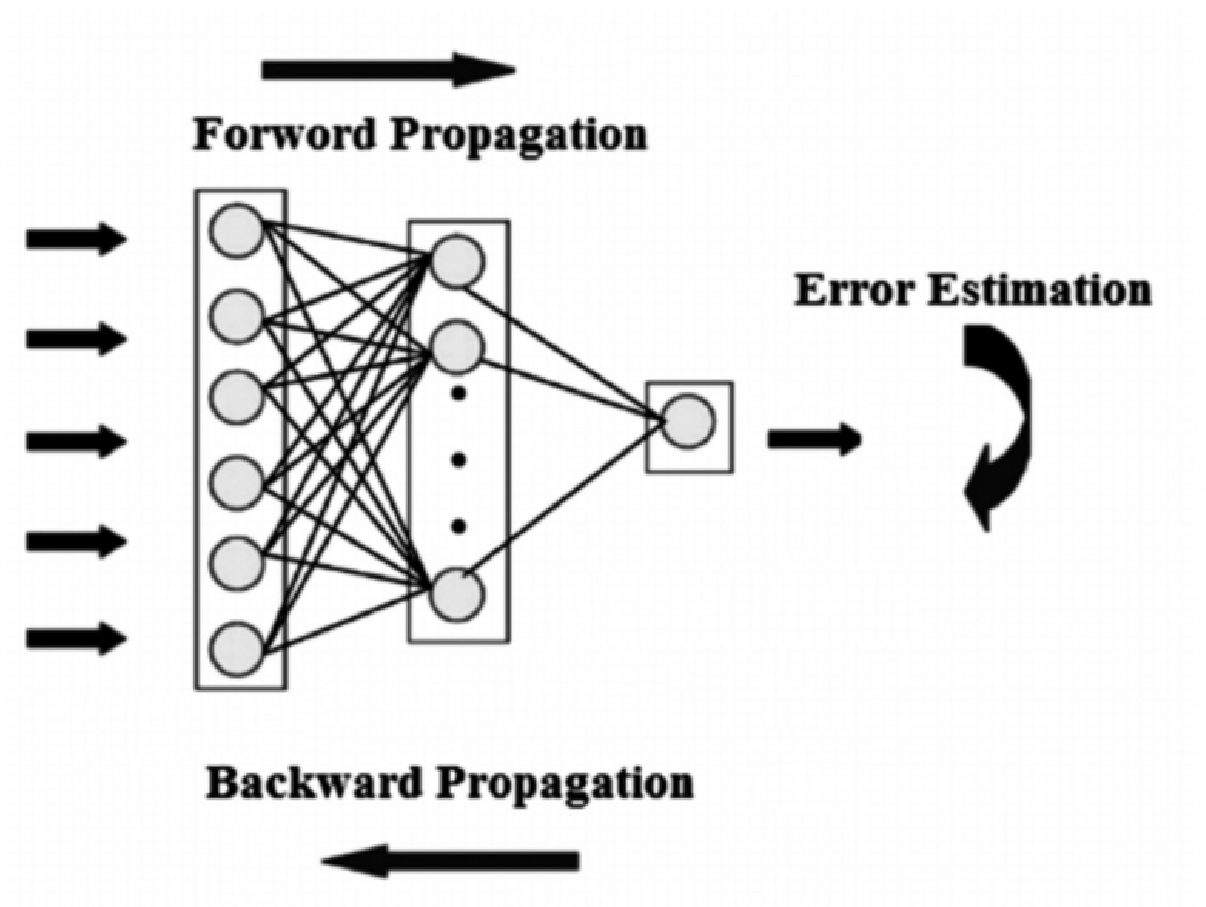
Machine learning's workhorse algorithm is backpropagation, sometimes known as BP or backprop. The gradient of an error function in relation to the weights of a neural network is calculated using backpropagation. To obtain the partial derivative of the mistakes in relation to the weights, the algorithm goes backwards through the gradient layers. Backprop then trains with smaller error margins by using the heavier weights.

It is common knowledge that forward propagation is used in neural networks to determine if a model's output is right or erroneous. When you use backpropagation, you go backwards through your neural network to determine the partial derivatives of the mistake with respect to the weights, and then subtract that value from the weights to get your final result.

Gradient descent, a technique that iteratively minimises a given function, uses their derivatives. When the mistake is reduced, the weights are increased or decreased accordingly. The training procedure is exactly how a neural network learns.

Backpropagation, on the other hand, entails experimenting with model weights while it is in motion.

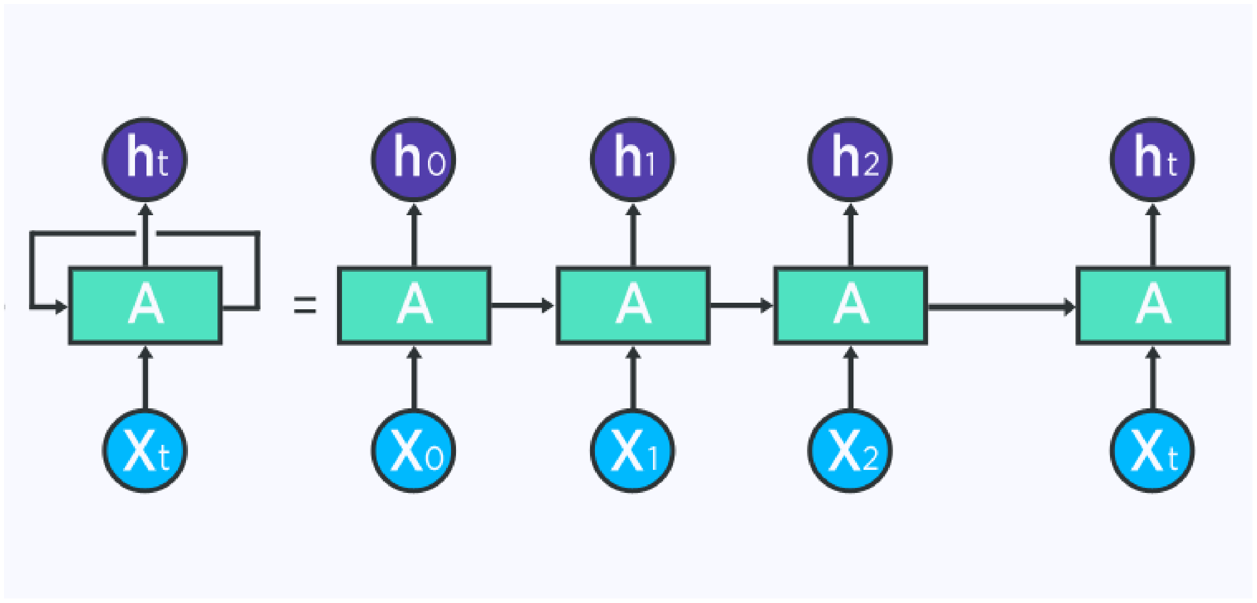
In a feed-forward neural network, forward propagation and backpropagation can be shown in the image below:



If you want to do backpropagation on an unrolled RNN, you use BPTT. With the help of Unrolling, you can see and understand what's happening in the network. Although backpropagation is usually handled automatically when using standard programming frameworks to create recurrent neural networks, understanding how it works is important for troubleshooting issues that may arise throughout the development process.

RNNs can be thought of as a series of backpropagation-trained neural networks.

An RNN is seen unrolled in the figure below. After the equal sign, the RNN is unrolled on the left side of the screen. Since the different time steps are visible and information is transmitted from one time step to the next, there is no cycle following the equal sign.. For the reasons stated above, an RNN can be thought of as nothing more than a collection of neural networks.



When using BPTT, the mistake is propagated backwards from the most recent timestep all the way to the earliest time step. Since the error can be calculated for each timestep, the weights can be updated as needed. Keep in mind that if you have a large number of timesteps, BPTT will be computationally expensive.

Two drawbacks of standard RNN's

However, in order to fully appreciate the challenges faced by RNNs, you must first have a basic understanding of gradients.

A gradient can be thought of as a partial derivative of its inputs. Just think of it like this: a gradient quantifies how much a function's output changes if its inputs are changed slightly.

Alternatively, you can think of a gradient in terms of a function's inclination angle. More gradient means steeper slope and more rapid learning for a model. A slope of zero means that the model has reached its learning limit. In order to calculate a gradient, all weights are compared to their error, and then the difference is calculated.

The drawbacks are described below:

1. Exploding gradients are seen when the algorithm gives excessive weights without any justification. Fortunately, truncating or squashing gradients is a simple solution to this issue.

2. When the gradient values are too small, the model stops learning or takes far too long as a result of disappearing gradients. These gradients were far more difficult to deal with than the explosive ones in the 1990s. Fortunately, Sepp Hochreiter and Juergen Schmidhuber came up with the concept of LSTM, which allowed them to find a solution.