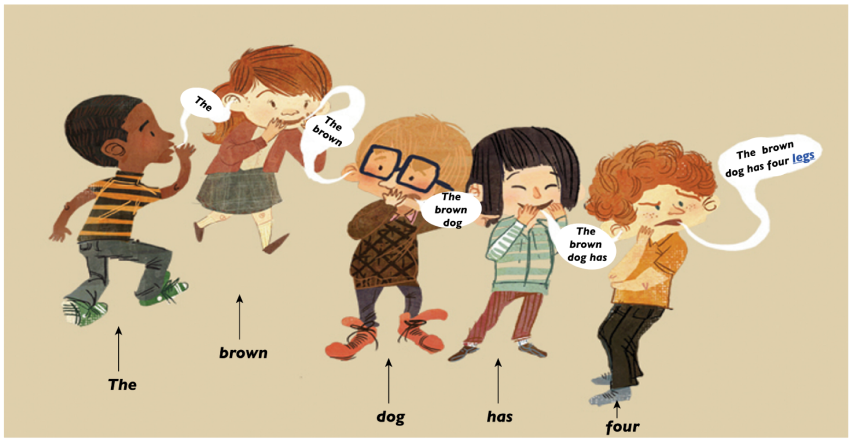
Eniola Alese Follow
I dont know so I learn and write.
Apr 13 · 5 min read

Understanding RNNs using the game of Chinese whispers

Recurrent Neural Networks (RNNs) are a popular variant of artificial neural networks which work really well on sequential data types i.e. a set of data points which are arranged in a particular order such that related data points follow each other. Some examples of sequential data types are stock market prices, audio and video data, DNA sequences, sensor data, natural language text, etc.

To gain intuition of how RNNs work, let's assume we are playing the popular children's game Chinese whispers and the objective of our game is for the last person to correctly predict the missing word in the sentence: "*The brown dog has four* _____"



playing Chinese whispers

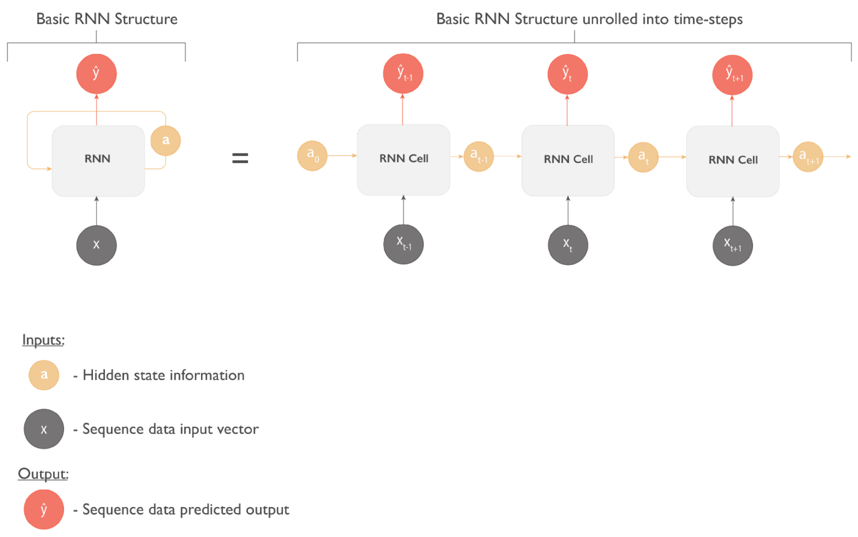
To achieve this the players form a line and are each given a word in the sentence according to how the word is arranged in the sentence (player 1 receives *"the"*, player 2 receives *"brown"*, player 3 receives *"dog"* and so on). The first person in the line whispers the word they have into the ear of the next person in the line, and then the next person whispers the word they heard from the previous person, together with the word they were also given to the next person in line and this goes on and on, until the last player is reached, who then tries to predict the missing word and announces the message to the entire group.

We can easily see that the last player would be able to correctly guess that the missing word is *"legs". This* is because he is going to be able to derive the word from the context of the earlier part of the sentence: *"The brown dog has four"* , that he just heard from the previous player.

The above illustration is **the basic principle of how RNNs work,** they are able to predict the next output of a sequence by taking information from the previous sequence and combining it with the input of the current sequence.

Unrolling RNNs

As we see from above, RNNs are capable of making predictions by repeating the same process across an entire sequence. The figure on the left below, shows the basic recurrent network structure; it takes in both the sequence data input vector (x) and the hidden state information (a) and uses them to predict the sequence data output.



To properly visualize how they operate, we usually unfold or unroll the RNN into a repetitive chain of cells or time-steps which correspond to the length of the sequence data (figure on the right above). For example, if the sequence we care about is a sentence of 3 words, the network would be unrolled into a 3 time-step RNN network, one time-step for each word.

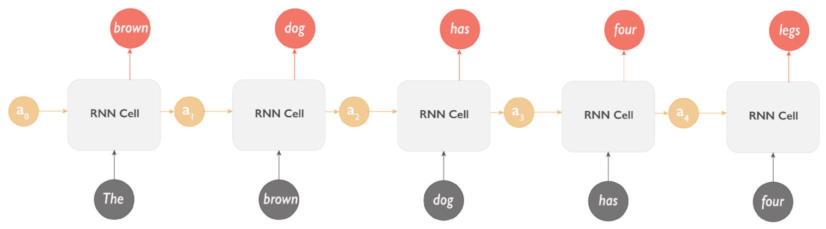
RNN Design Architectures

The input and output of sequential data can come in various forms and lengths. For example in a sentiment analysis task, the input is usually a sequence of text and the desired output can be an integer (1–5 rating scale) or a single text (good/bad, positive/negative/neutral) while in a speech-text task we can have the input as a sequence of acoustic signals from a sound wave and the output as also a sequence of corresponding text.

Because of these variations, the input and output structure of RNNs can also be designed to match the sequential task to be solved. Some of the common architecture types are:

1. Many-to-many architecture (same sequence length)

This is the basic RNN structure where the number of input sequence is the same as the number of output sequence at every time-step. An example application that uses this architecture is a **text generator** that predicts the next most likely word in a sentence, given the previous words.

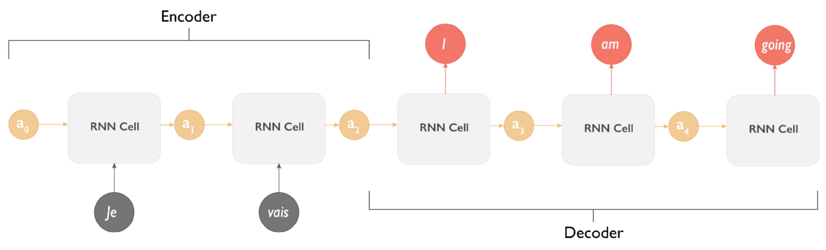


text generator using a many-to-many RNN architecture

In the image above, the text generator's input is a sequence of words "*The brown dog has four*" and its output is also a sequence of predicted next words "*brown dog has four legs.*"

2. Many-to-many architecture (different sequence length)

Another variation of the many-to-many architecture is in cases where the input and output both have different sequence lengths. An example application is in **machine translation** tasks, where the input is a sequence of words in a source language (e.g. French) and the output is a sequence of words in a target language (e.g English). This architecture has two distinct parts; an **encoder** which takes in the input sentence, maps it into an internal state representation and then passes this representation to a **decoder** which then uses it to generate the output sentence.

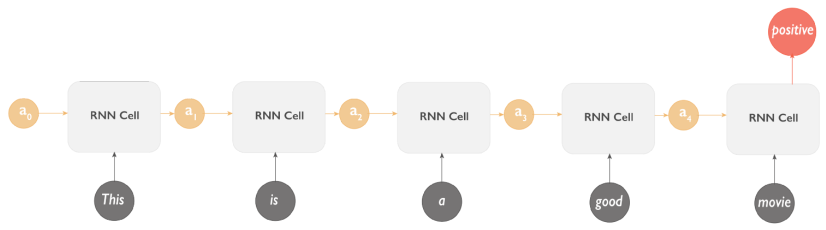


machine translation using many-to-many RNN architecture

In this architecture the decoder can only start predicting the output sequence after the encoder has processed the complete input sequence, unlike in the same sequence length architecture which starts predicting the output sequence immediately after each input sequence

3. Many-to-one architecture

In this architecture the RNN has a sequence of inputs at each time step but only outputs a single value at the last time step. An example application that is a **sentiment analysis** task, where the objective is to classify a given input statement as having either a positive or negative sentiment.



sentiment analysis task using a many-to-one RNN architecture

In the sentiment analysis task above, rather than having an output at every single time-step, the RNN reads in the entire sequence of words "*This is a good movie*" and outputs just a single value "*positive*" at the last time step.

4. One-to-many architecture

Here the RNN takes in a single input at the first time step and outputs a sequence of values at the remaining time steps. In this architecture , some applications often also take the predicted output at each time step and feed it into the next layer as an input value. An example application for this architecture is in **image captioning**, here the RNN takes an image as its input and outputs a sequence of words describing what is going on in the image.

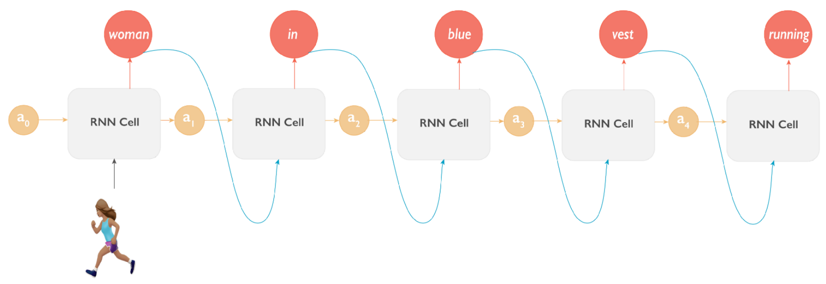


image captioning using a one-to-many RNN architecture

In the above, the input is an emoji of a woman running and the output is a sequence of predicted words that describe the image "*woman in blue vest running*".

Conclusion

In this post we covered a brief introduction to recurrent networks, various types of architectures and example applications. In the next post in the recurrent network series, we would dive into RNN equations, forward propagation and understanding Back Propagation Through Time (BPTT).