

# The RNN API

Sequences present several complexities.

Let's begin by better understanding functions that have sequences as inputs and output.

We call this the [RNN API \(RNN\\_API.ipynb\)](#).

## Inside an RNN layer

By now you hopefully have a good intuitive understanding of a Recurrent Layer, but lack the details

Let's open up the hood and [go inside an RNN \(RNN Workings.ipynb\)](#).

# RNN in action

A concrete example may help you to appreciate the power of an RNN.

Let's see an [RNN in action](#) ([RNN in action.ipynb](#)).

# What is *really* going on inside an RNN

At this point

- You appreciate the ability of an RNN to operate on sequences
- Understand the mechanics of the internal workings

But the update equations don't really convey an intuition about *how* the RNN achieves its power.

Let's try to visualize the latent state of an RNN in order to get a better grasp.

[RNN Visualization \(RNN\\_Visualization.ipynb\)](#).

# **RNN practicalities**

## **Sequences: Variable length**

There are lots of small potholes one encounters with sequences.

What if the examples of my training set have widely varying lengths ?

- Within a batch, short examples may behave differently than long examples:
  - Maybe learn less in short examples, noisier gradient updates
- Padding sequences to make them equal length
  - Pad at the start ? Or at the end ?

The general advice is to arrange your data so that an epoch contains examples of similar lengths.

- You may require multiple fittings, one per length

# Issues with RNN's

Although an RNN layer seems powerful (and a little magical) we have glossed over some big issues

- Can they handle *long* sequences or are they subject to "forgetting" ?
  - Short term versus long term memory trade offs
- Can we really unroll a computation over a long sequence ?
  - Gradient computation potentially more difficult in very deep graphs
- What are the practical difficulties in Keras with long sequences



These will be the topics of subsequent modules.

- Some topics require an in-depth understanding of Gradient Computation (still to come !)

# Conclusion

The Recurrent layer was yet another layer type that we have introduced in rapid succession.

We chose to do this as a "sprint" rather than a "marathon" so that you can start coding and experimenting.

Use the opportunity ! This is where the real learning will happen.

Our next topics will be a more in-depth exploration of issues that may not have come into view during the sprint.

In [2]: `print("Done")`

Done