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)](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.

The task we solve is called Sentiment Analysis

- Given a sequence of words
- Is the sentiment express Positive or Negative ?

In particular

- The examples are movie reviews from IMdB

IMdb examples

This notebook is from the (future) module on NLP.

The beginning of this notebook addresses the aspects particular to dealing with Natural Language

- [the input examples (Keras_examples_imdb_cnn.ipynb#Examine-the-text-data)](Keras_examples_imdb_cnn.ipynb#Examine-the-text-data)
- the pre-processing steps necessary for NLP:
    - breaking character strings into words
    - mapping words to integers (index within a finite vocabulary)

To summarize our approach to dealing with words

- We have a finite vocabulary $\mathbf{V}$
- Words are One Hot Encoded
- So the input sequence $\mathbf{x}_{(1)} \ldots \mathbf{x}_{(T)}$ is
    - a sequence of length T
    - of OHE vectors of length $||\mathbf{V}||$

We also have the option of transforming the OHE word encodings into *Embeddings*

- Embeddings are a transformation of the OHE into shorter vectors that *also* encode some "meaning" to words
- Our initial pass: we just use an Identity transformation
- This will be covered in detail in the NLP module.

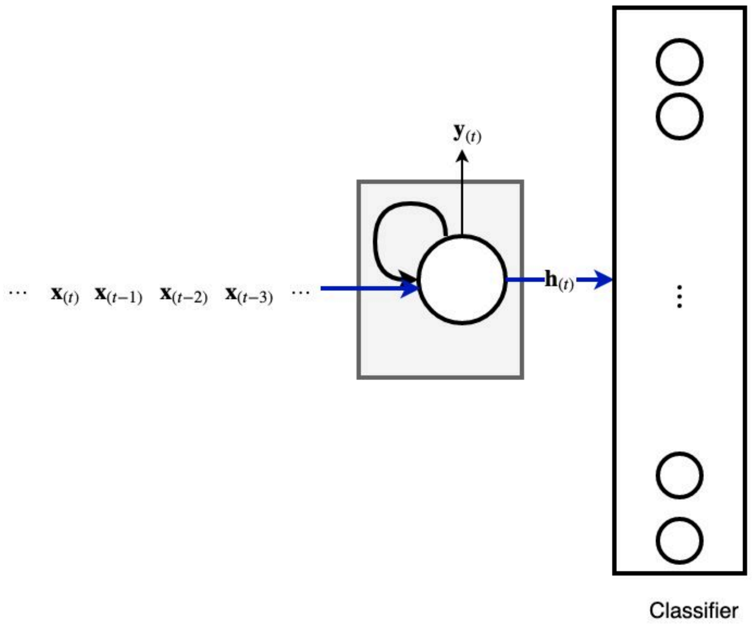
We will cover the NLP aspects in more detail in the NLP module.

For now: we will focus on using an RNN to create a finite length representation of the (potentially unbouncded length) sequence of words.

The model we create follows the paradigm in the introduction

- Using an RNN to create a fixed length encoding of a variable length sequence
- A Head Layer that is a Binary Classifier

RNN Many to one; followed by classifier



Let's look at the code. The main take-away

- the RNN is a layer just like any other
- we use it as the first layer in a Sequential model
    - to reduce the sequence to a finite vector
- once we have a finite vector, `Dense` layers further transform the representation
- until we have a final binary Classifier "head"

[RNN(LSTM) model (Keras_examples_imdb_cnn.ipynb#LSTM-w/o-One-Hot-Encoding-the-input:-what-happens-?)](Keras_examples_imdb_cnn.ipynb#LSTM-w/o-One-Hot-Encoding-the-input:-what-happens-?)

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

Long sequences

We will learn that long sequences present a challenge to training RNN's

- vanishing gradients
- back propagation of gradients takes a long time

There is also the practical matter of long sequences (e.g., greater than the "max" length allocated to a variable).

A Deeper Dive deals with the practicl treatment of long sequences (RNN_Long_Sequences.ipynb)

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
```