RNN as a layer

During one time step t, the RNN

- Takes element t of $\mathbf{x}$ as input: $\mathbf{x}_{(t)}$
- Computes a new latent state $\mathbf{h}_{(t)}$
- Optionally outputs element t of the output: $\mathbf{y}_{(t)}$

$$\mathbf{h}_{(t)}, \mathbf{y}_{(t)} = f(\mathbf{x}_{(t)}; \mathbf{h}_{(t-1)})$$

$\mathbf{h}_{(t)}$ is used in the next time step of the RNN but may not be externally visible.

Let's describe the inputs/outputs of an RNN layer from the perspective of what is and is not visible.

If we draw a box around the unrolled RNN, we can see the "API":

RNN many to one API

- The input sequence $\mathbf{x}$ of length T is depicted as coming from below
- The output of the layer is $\mathbf{y}_{(T)}$
- Everything inside the box is *not visible*
- Until the entire sequence $\mathbf{x}$ has been processed

- Output $\mathbf{y}$ is available to be fed to another layer (i.e., not the same RNN layer)
- Latent state $\mathbf{h}$ is retained by the RNN layer

Many to one

The above API was for an RNN layer computing a many to one function

- Sequence input, single vector as output

A many to one mapping is particularly useful

- If one considers $\mathbf{y}_{(T)}$ a fixed length summary of variable length sequence $[\mathbf{x}_{(1)}, \ldots, x_{(T)}]$
- Which is amenable for processing by a layer requiring a fixed length input

Many to many

We can show the API for an RNN layer computing a many to many function

- Sequence input, sequence of vector output

Essentially, the "internal" (inside the box) workings are exposed to the user, rather than hidden.

RNN many to many API

In order to get Keras to implement the many to many API, optional arguments are used when constructing the layer

- `return_sequences`
- `return_states`
- both default to `False` in Keras.

These control whether the RNN layer returns a sequence

$$[\mathbf{h}_{(1)}, \ldots, \mathbf{h}_{(T)}]$$

$$[\mathbf{y}_{(1)}, \ldots, \mathbf{y}_{(T)}]$$

or just

$$\mathbf{h}_{(T)}$$

$$\mathbf{y}_{(T)}$$

One to many

It may seem strange to generate a sequence output from a single input, but consider

- Feeding the output of step $(t-1)$ as *input* to step $t > 1$

$$\mathbf{x}_{(t)} = \mathbf{y}_{(t-1)}$$

A picture should help

RNN one to many API

This will be particularly useful when the outputs $\mathbf{y}_{(t)}$ have an element of randomness

- A new output sequence is generated even when the same input "seed" $\mathbf{x}$ is used

We will show how an architecture like this can be used to *generate*

- A story (sequence of words)
- From a single (or small length sequence) "seed" word

Combining RNN layers

There are some typical paradigms in which layers are combined.

Stacked RNN layers

By feeding the output sequence into another RNN layer, we can achieve stacked layers

RNN Stacked layers

Encoder/Decoder architecture

An Encoder/Decoder architecture has

- An Encoder RNN layer, implementing a many to one relationship
- Followed by a Decoder RNN layer, implementing a one to many relationship

RNN Encoder/Decoder

- The input sequence $\left[\mathbf{x}_{(1)} \ldots \mathbf{x}_{(\bar{T})}\right]$
- Is summarized by $\bar{\mathbf{h}}_{(\bar{T})}$, the final latent state of the Encoder RNN
- Which is used to seed the Decoder RNN
- Producing new sequence $\left[\hat{\mathbf{y}}_{(1)} \ldots \hat{\mathbf{y}}_{(T)}\right]$

Note that T is not neccesarily equal to $\bar{T}$

- The Decoder is seeded by a singleton
- So the output length T is no longer dependent on the length $\bar{T}$ of input $\mathbf{x}$
- Language translation: not necessarily a one-to-one correspondence between word t of each language

Recall that $\bar{\mathbf{h}}_{(\bar{t}\,)}$ is a fixed length encoding of the input prefix $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(\bar{t}\,)}$

So $\bar{\mathbf{h}}_{(\bar{T})}$, which initializes the Decoder, is a summary of the entire input sequence $\mathbf{x}$.

This fact enables us to decouple the Encoder from the Decoder

- The consumption of input $\mathbf{x}$ and product of output $\hat{\mathbf{y}}$ do not have to be synchronized
- Allowing for the possibility that $T \neq \bar{T}$

The combination of the two is used to solve a class of problems called *Sequence to Sequence*

- Transform one sequence to another
- Language translation: sequence of English words to sequence of Mandarin symbols
- Captioning: sequence of image frames to sequence of words describing the movie

Conclusion

We explained how an RNN may compute several types of relationships

- Many to one
- Many to many
- One to many

This variety arises because both input and output may be sequences.

Sequence to Sequence problems (a variant of "many to many") is a particularly important class of problems that can be solved with RNN's.

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