The Long Short Term Memory (LSTM) layer

The "vanilla" Recurrent Neural Network (RNN) layer that we introduced was powerful but somewhat limited

- It suffered from gradients that vanished or exploded
- It's memory tended to be short-term
- Unable to capture dependencies that were too far separated in time

Researchers developed advanced Recurrent Network layer types to address these specific issues.

The *Long Short Term Memory* (LSTM) layer is one such example.

The importance of selectively forgetting

An RNN layer, at time step t

- Takes input element $\mathbf{x}_{(t)}$
- Updates latent state $\mathbf{h}_{(t)}$
- Optionally outputs $\mathbf{y}_{(t)}$

according to the equations

$$
\begin{aligned}
\mathbf{h}_{(t)} &= \phi(\mathbf{W}_{xh}\mathbf{x}_{(t)} + \mathbf{W}_{hh}\mathbf{h}_{(t-1)} + \mathbf{b}_h) \\
\mathbf{y}_{(t)} &= \mathbf{W}_{hy}\mathbf{h}_{(t)} + \mathbf{b}_y
\end{aligned}
$$

where

- ϕ is an activation function (usually $\tanh$)
- $\mathbf{W}$ are the weights of the RNN layer
    - partitioned into $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$, $\mathbf{W}_{hy}$
    - $\mathbf{W}_{xh}$: weights that update $\mathbf{h}_{(t)}$ based on $\mathbf{x}_{(t)}$
    - $\mathbf{W}_{hh}$: weights that update $\mathbf{h}_{(t)}$ based on $\mathbf{h}_{(t-1)}$
    - $\mathbf{W}_{hy}$: weights that update $\mathbf{y}_{(t)}$ based on $\mathbf{h}_{(t)}$

RNN

Latent state $h_{(t)}$ is

- A *fixed length* encoding of the variable length input sequence $[\mathbf{x}_{(1)} \ldots \mathbf{x}_{(t)}]$
- All essential information about the prefix of $\mathbf{x}$ ending at step t is recorded in $\mathbf{h}_{(t)}$

$\mathbf{h}_{(t)}$ is charged with several tasks

- Determining the time t output $\mathbf{y}_{(t)}$ (for a many to many RNN)
- Determining the next latent state $\mathbf{h}_{(t+1)}$

But is each and every element $\mathbf{x}_{(t')}$ needed for both these tasks ?

Probably not.

Consider sequence $\mathbf{x}$ as frames in a movie.

- Is every detail of the early scenes of a movie
- Relevant to the final scene ?

It would be very powerful

- To be able to "forget" synthetic features that are **no longer** relevant
- And *selectively* update **immediately relevant** features
- While leaving unchanged those features needed **far in the future**

This is where the "gates" (`if` , `switch/case`) of Neural Programming are relevant.

They will be used in an LSTM

- To determine ("gate") when an individual feature in latent state $\mathbf{h}_{(t)}$ "forgets" (reset)
- To determine which individual features in latent state $\mathbf{h}_{(t)}$ get updated at step t

Conclusion

The LSTM is an advanced Recurrent Neural Network (RNN) layer type.

The "gating" mechanism of Neural Programming will be key.

It will endow the LSTM with

- The ability to forget a feature
- Selectively update other features

This will allow the LSTM to mitigate the drawbacks of "vanilla" RNN layers

- Short term memory
- Vanishing/Exploding gradients

In [2]:

```
print("Done")
```

Done