# Visualization of RNN hidden state

It's nice that we have update equations telling us the mechanics of an RNN layer.

But *what* is an RNN layer *really* doing ? How does it make the magic happen ?

One plausible theory is that

- The individual elements of the latent state $\mathbf{h}$
- Are acting like *counters*
- Incrementing/Decrementing according to the input

A visualization can confirm this theory (in some cases).

- Pick one element $\mathbf{h}_j$ of the latent state
- Examine the sequence $[\mathbf{h}_{(t),j} | 1 \leq t \leq T]$ of this element
- Correlate changes in $\mathbf{h}_{(t),j}$ with the input sequence $[\mathbf{x}_{(t)} | 1 \leq t \leq T]$

Below is a visualization (http://karpathy.github.io/2015/05/21/rnn-effectiveness/#visualizing-the-predictions-and-the-neuron-firings-in-the-rnn)

- Of several elements of the hidden state
- Where the value of the element is color-coded
    - Red: High; Blue: Low
- And overlaid on the corresponding element of $\mathbf{x}_{(t)}$
- On an RNN trained on a "predict the next character" in the sequence task

Here is an element ("cell") that becomes active inside "bracketed text"

- Inside quotes (" .. ")
- Inside code comments (/ ... /)

State activations after seeing prefix of input

Here is a cell that seems to be

- Counting the *depth* of nesting of code

**State activations after seeing prefix of input**

And here is a cell that has been interpreted

- As predicting end-of-line characters

**State activations after seeing prefix of input**

Of course, this is a matter of interpretation rather than mathematics.

Still: there is some logic in believing that counters

- Can capture structure
- Sufficient to encode the probability of the next character (our target)

In a later module

- We will study a more advanced Recurrent layer called an LSTM
- It's internal workings are closely aligned with the notion of implementing counters

In [2]:

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

```
Done
```