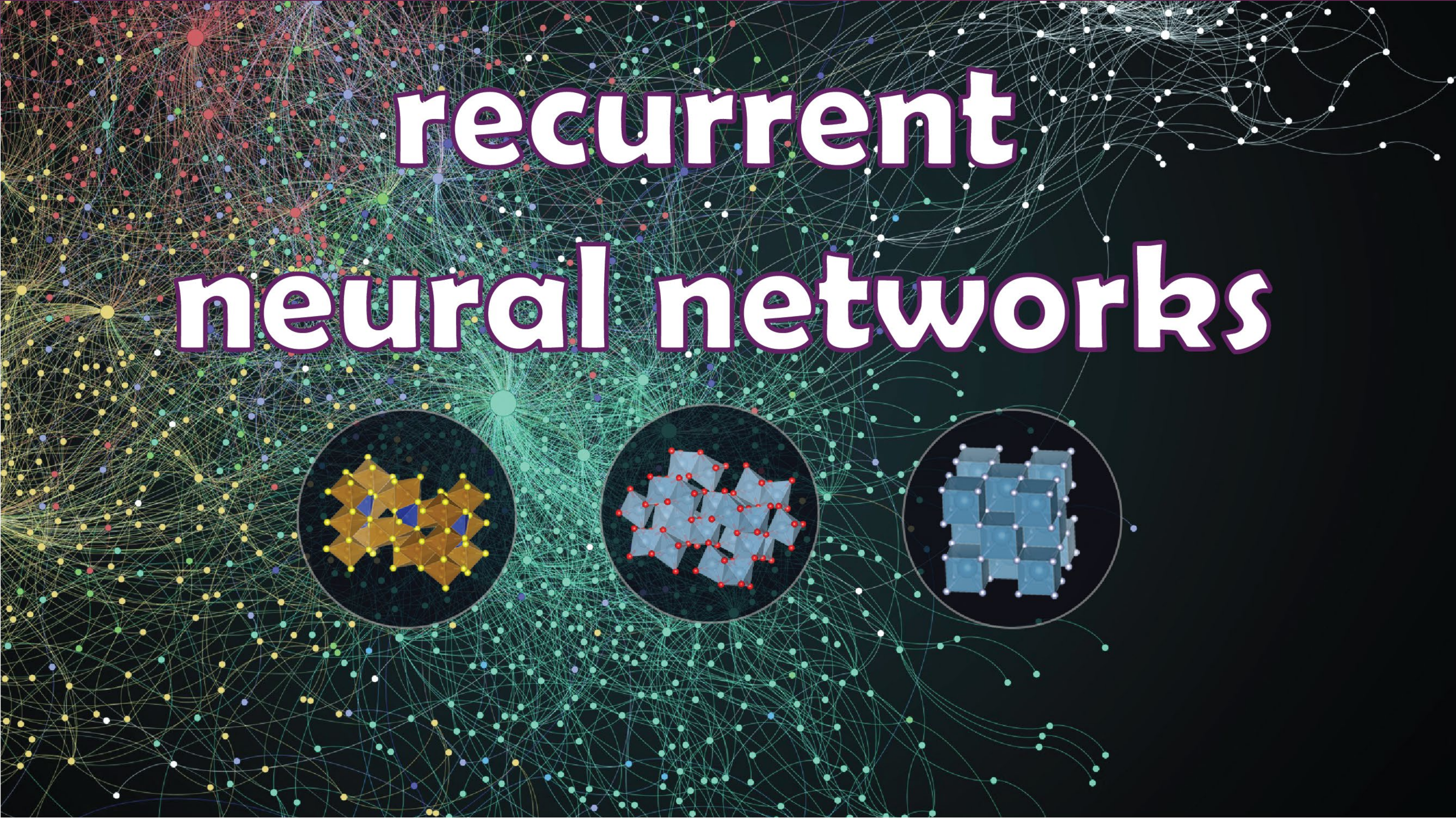
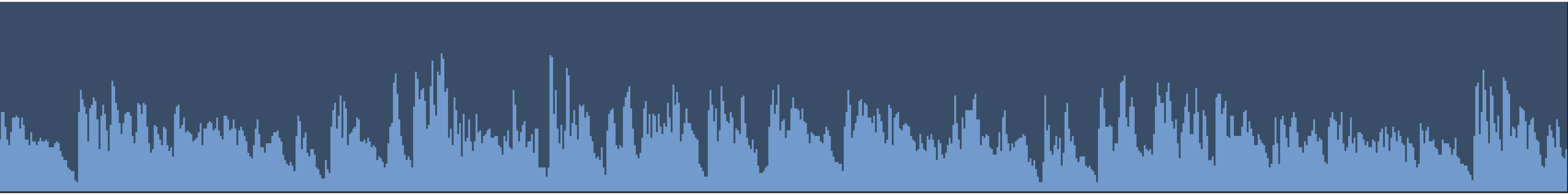
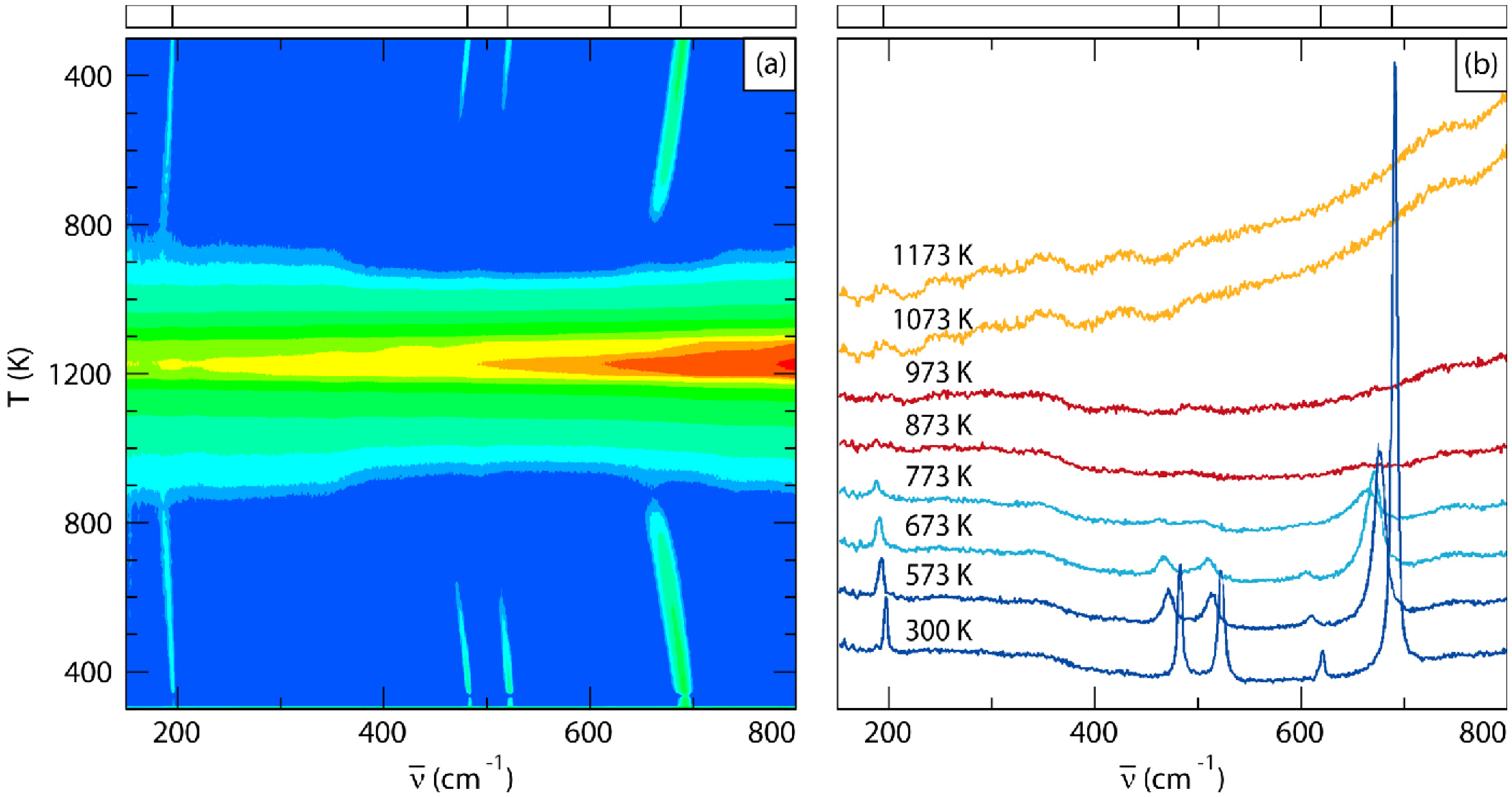

recurrent
neural networks

Recurrent neural networks were made for sequential datasets



In this paper we investigate the high-temperature structure of Co_3O_4, a compound that has been studied extensively over the last 60 years due to its unresolved high-temperature structure. In situ thermal analysis and x-ray diffraction confirm previously reported high-temperature structural changes and show that these changes are unrelated to the high-temperature decomposition to CoO. Raman-active peaks are also extinguished over the same temperature range. By considering the changing lattice parameter, A-O, and B-O bond lengths as well as cation size we are able to calculate the degree of inversion which reaches a maximum of 0.6. To further study the structure in this experimentally inaccessible range we quench samples and perform ex situ measurements including redox titration, x-ray photoelectron spectroscopy, and neutron diffraction. We do not observe any evidence of large oxygen vacancy concentrations or octahedral Co^{3+}_B ions with high spin state. However, we do show an evolution in the magnetic moment from magnetic structure refinement from (2.4 μB) to (2.7 μB) that coincides exactly with the high-temperature anomaly and suggests partial inversion (0.46) of the spinel structure in fairly good agreement with the inversion calculated from bond lengths. Collapse

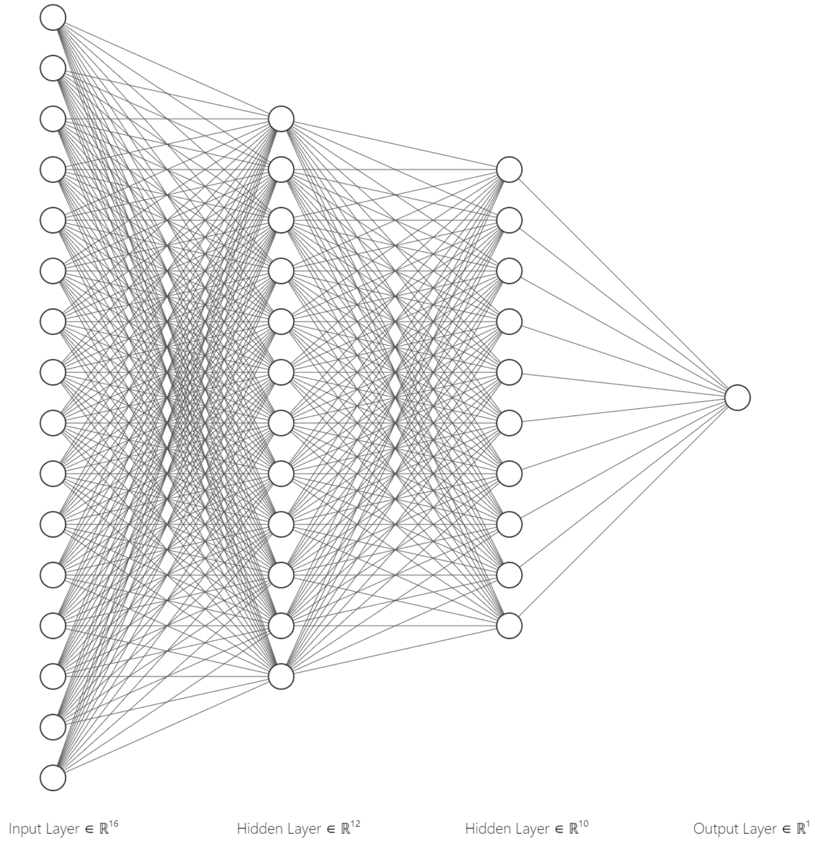
Recurrent neural networks were made for sequential datasets



Sparks et al, Phys Rev B., (2019)

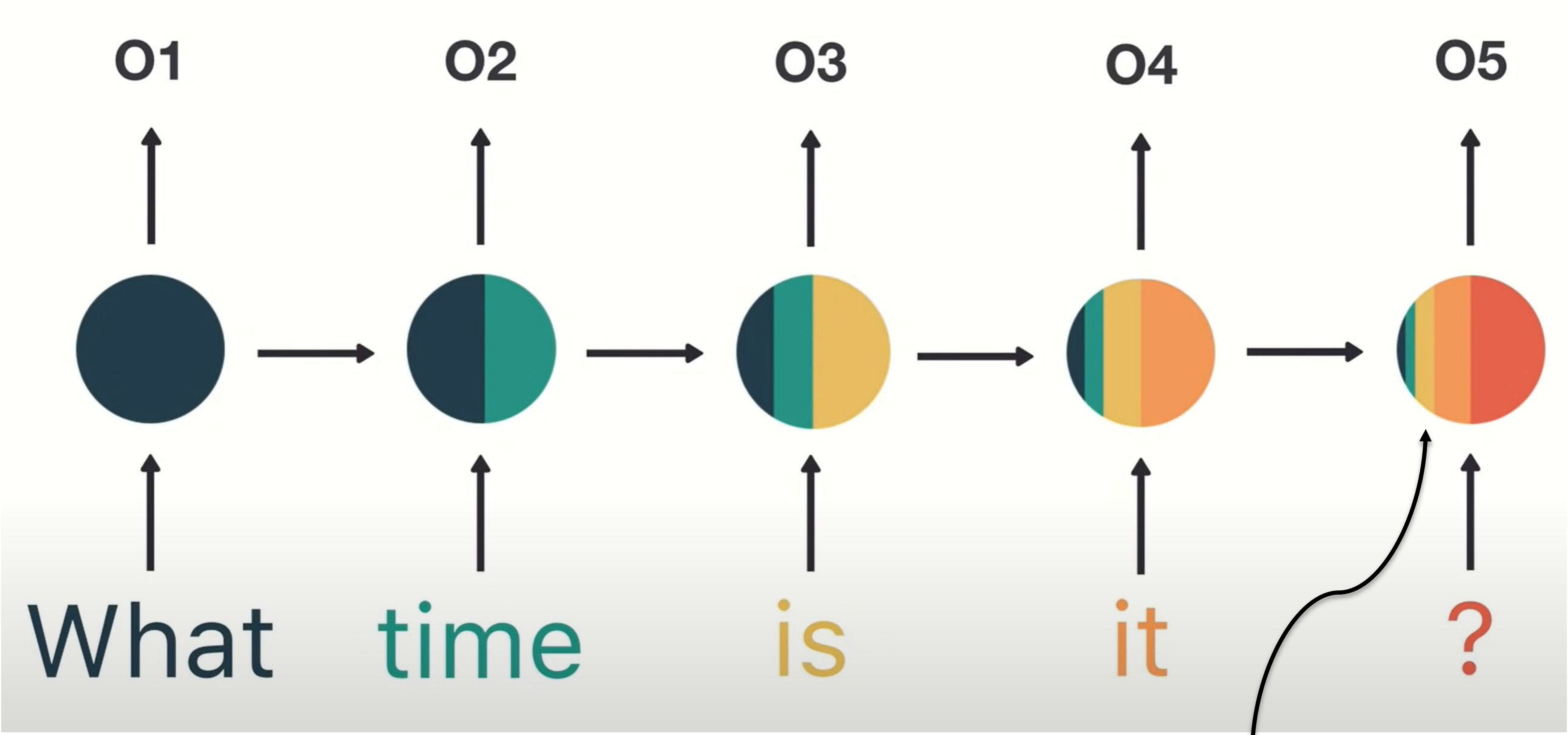
Normal feed-forward “vanilla” neural nets assume all data points are independent

There is no way to pass information forward from one sample to next



We would need a way to store “memory” of information from previous layer in building a subsequent layer

Each sequential data is fed through neural network, but we include hidden states



Short term memory due to vanishing gradient problem

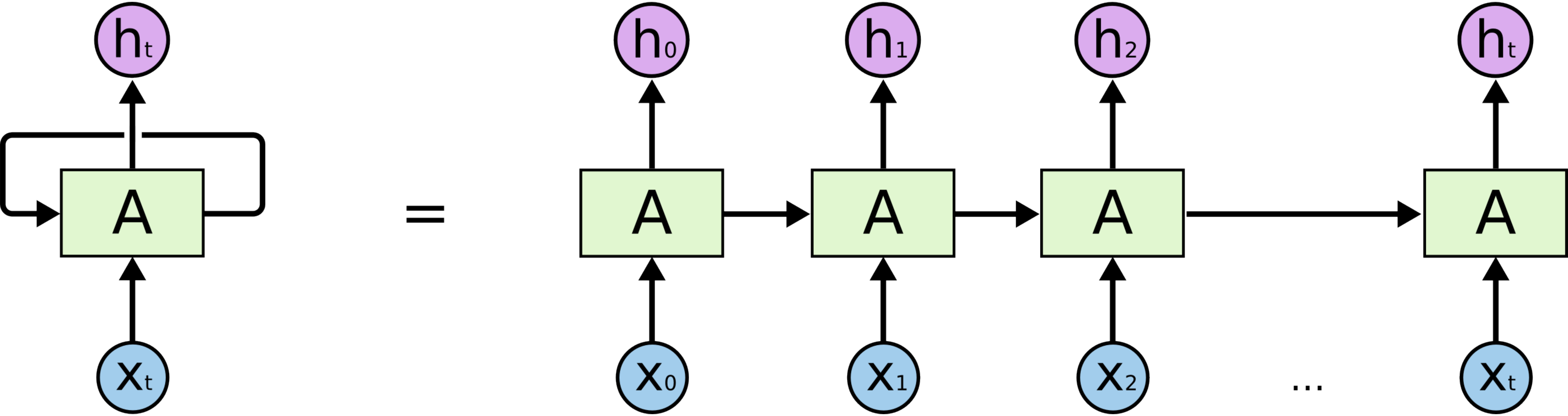
Michael Phi, Illustrated Guide to Recurrent Neural Networks: Understanding the Intuition, YouTube

Each sequential data is fed through neural network, but we include hidden states

```
rnn = RNN()
ff = FeedForwardNN()
hidden_state =[0.0, 0.0, 0.0, 0.0]

for word in input:
    output, hidden_state = rnn(word, hidden_state)

prediction = ff(output)
```

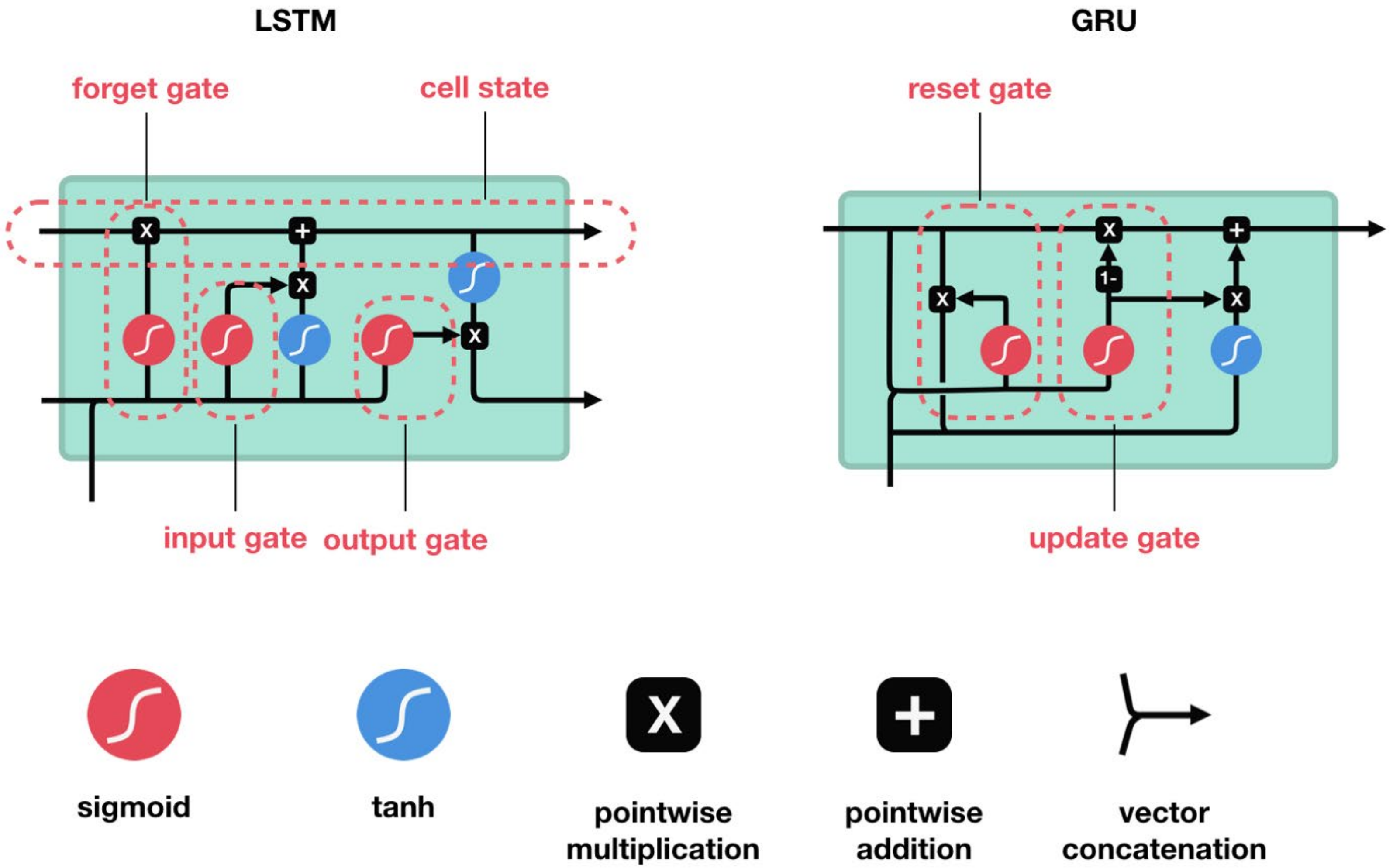
Michael Phi, Illustrated Guide to Recurrent Neural Networks: Understanding the Intuition, YouTube

Sequential memory is needed…. But short term memory is still a major problem



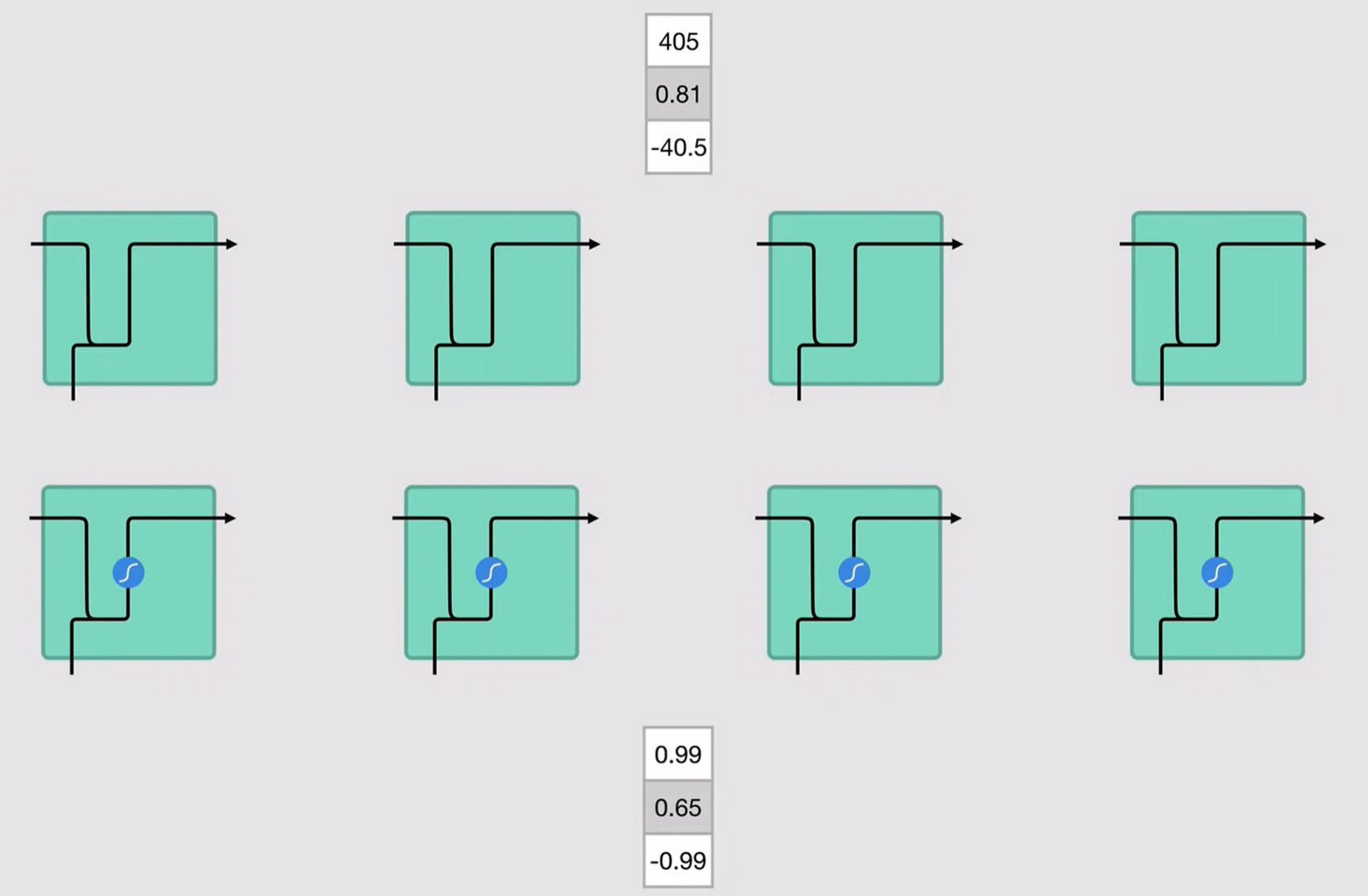
Solutions to the problem of short term memory include
LSTM (long short term memory) neural networks and
GRU (gated recurrent unit) neural networks
Regular RNNs are much faster and work where long memory isn't needed

LSTM and GRU's incorporate "gates" to help regulate the flow of information



Michael Phi, Illustrated Guide to LSTM's and GRU's: A step by step explanation, YouTube

A normal RNN uses tanh function to keep input + hidden state values from blowing up



Michael Phi, Illustrated Guide to LSTM's and GRU's: A step by step explanation, YouTube

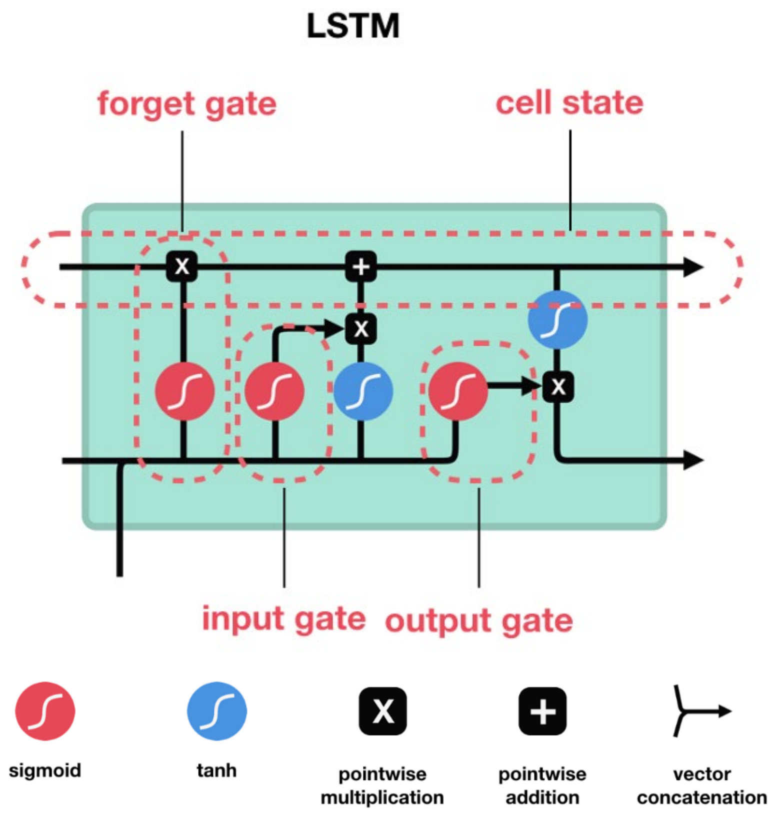
LSTM has many more operations in each neural network!

Cell state lets us hang onto and transfer useful into directly onto the next sequence chain (from first sequence all the way to end!)

Gates move information onto and off of cell state (sigmoid lets us easily make values zero to forget, 1 to keep in memory)

Input gate updates cell state using previous hidden state and current state (The sigmoid tells us what to keep from the tanh output)

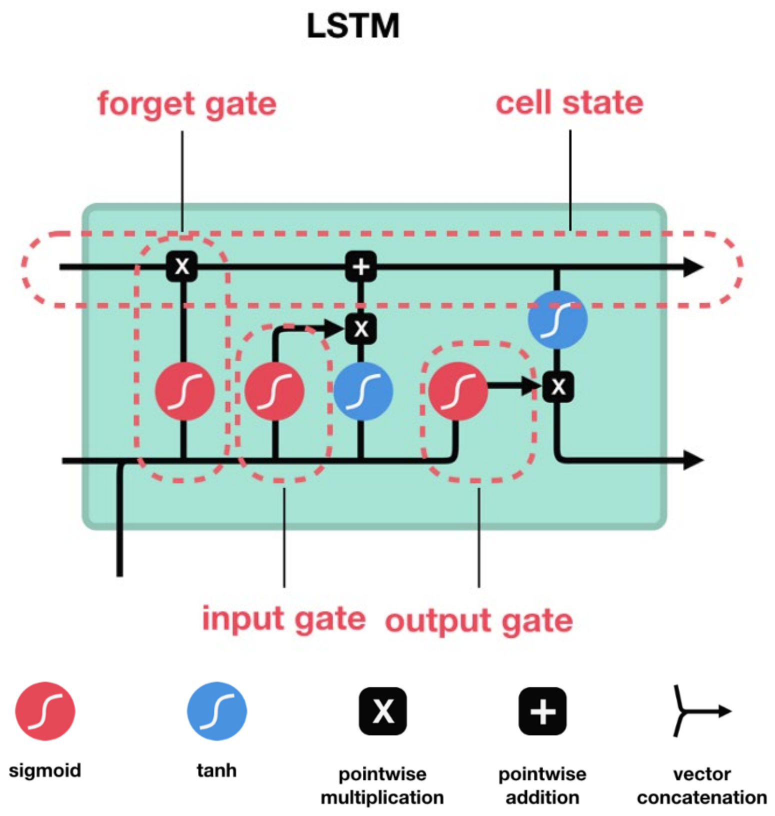
Output gate decides what the next hidden state should be. (tanh brings in current memory, sigmoid tells us what to keep or forget)



Michael Phi, Illustrated Guide to LSTM's and GRU's: A step by step explanation, YouTube

Or in pseudocode

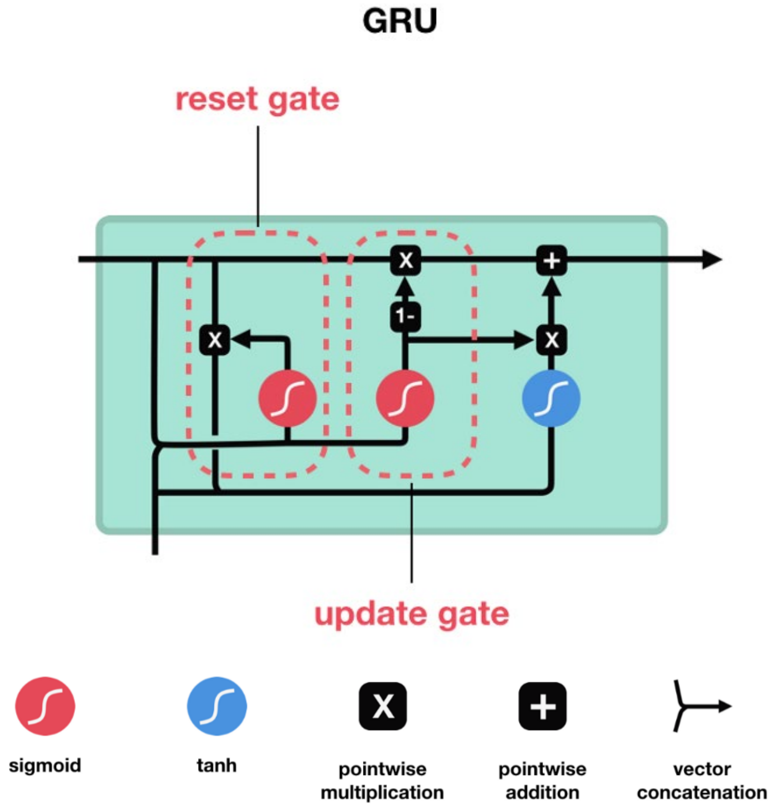
```
def LSTMCELL(prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)
    Ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(Ct)
    return ht, Ct


ct = [0, 0, 0]
ht = [0, 0, 0]

for input in inputs:
    ct, ht = LSTMCELL(ct, ht, input)
```



Michael Phi, Illustrated Guide to LSTM's and GRU's: A step by step explanation, YouTube

GRUs work in a similar way to an LSTM, but removed the cell state

Hidden state is used to hold all the memory
Update gate acts like the forget and input gate of LSTM.
Reset gate decides how much of past information should be forgotten



GRU's have fewer tensor operations so they train a bit faster than LSTMs

Michael Phi, Illustrated Guide to LSTM's and GRU's: A step by step explanation, YouTube

Let's go through the math of an LSTM

Forget gate

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Input gate

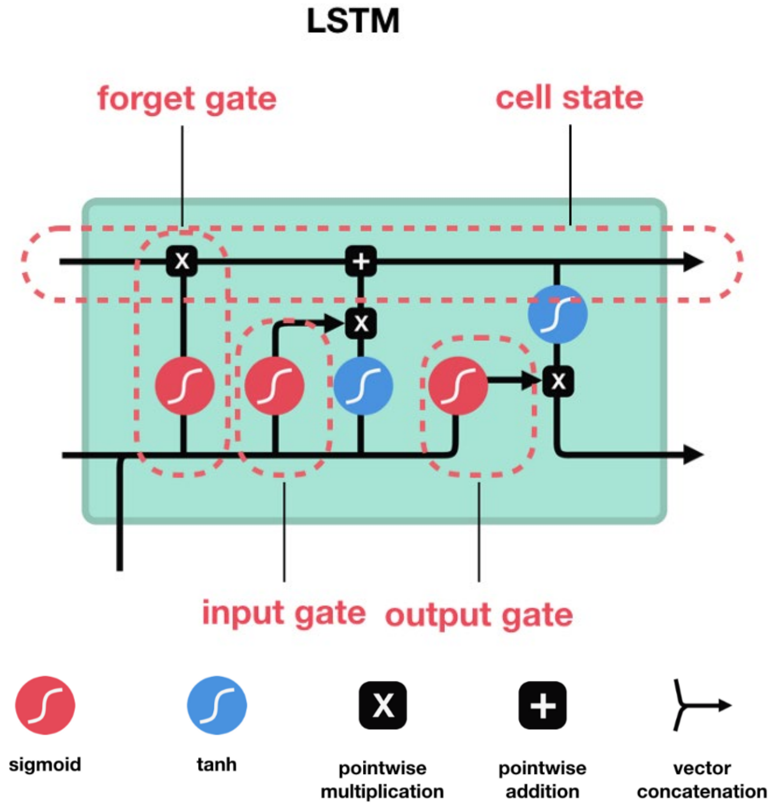
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Update cell state

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

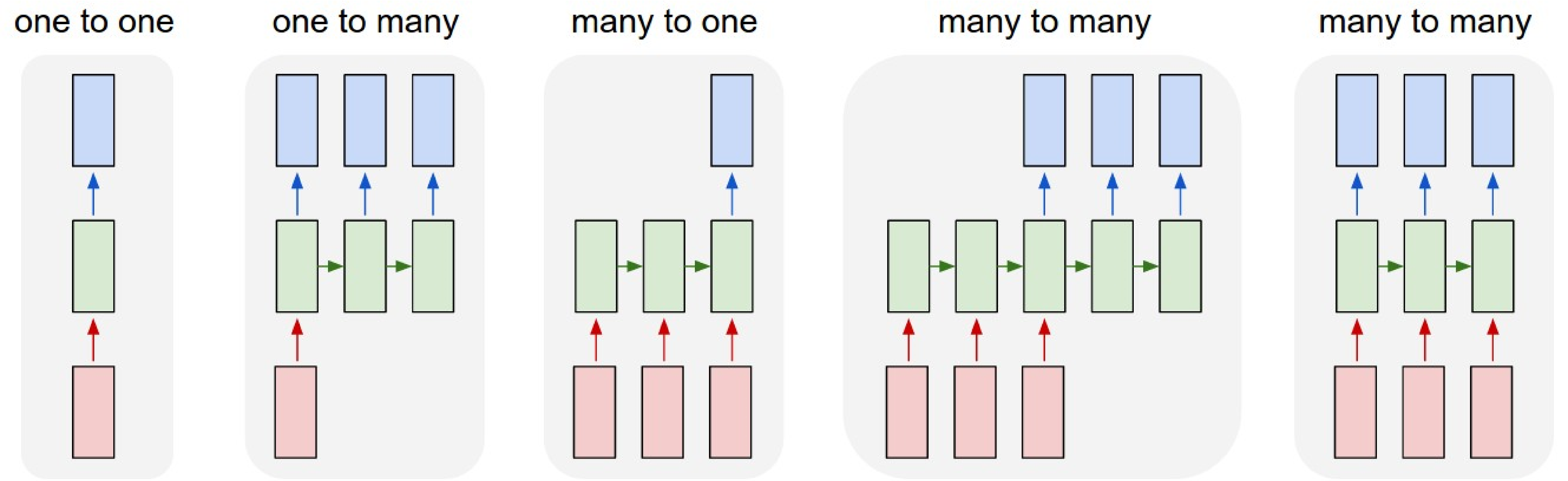
Output gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

There are more than one way to connect time series….



https://iq.opengenus.org/types-of-rnn/


Transformers