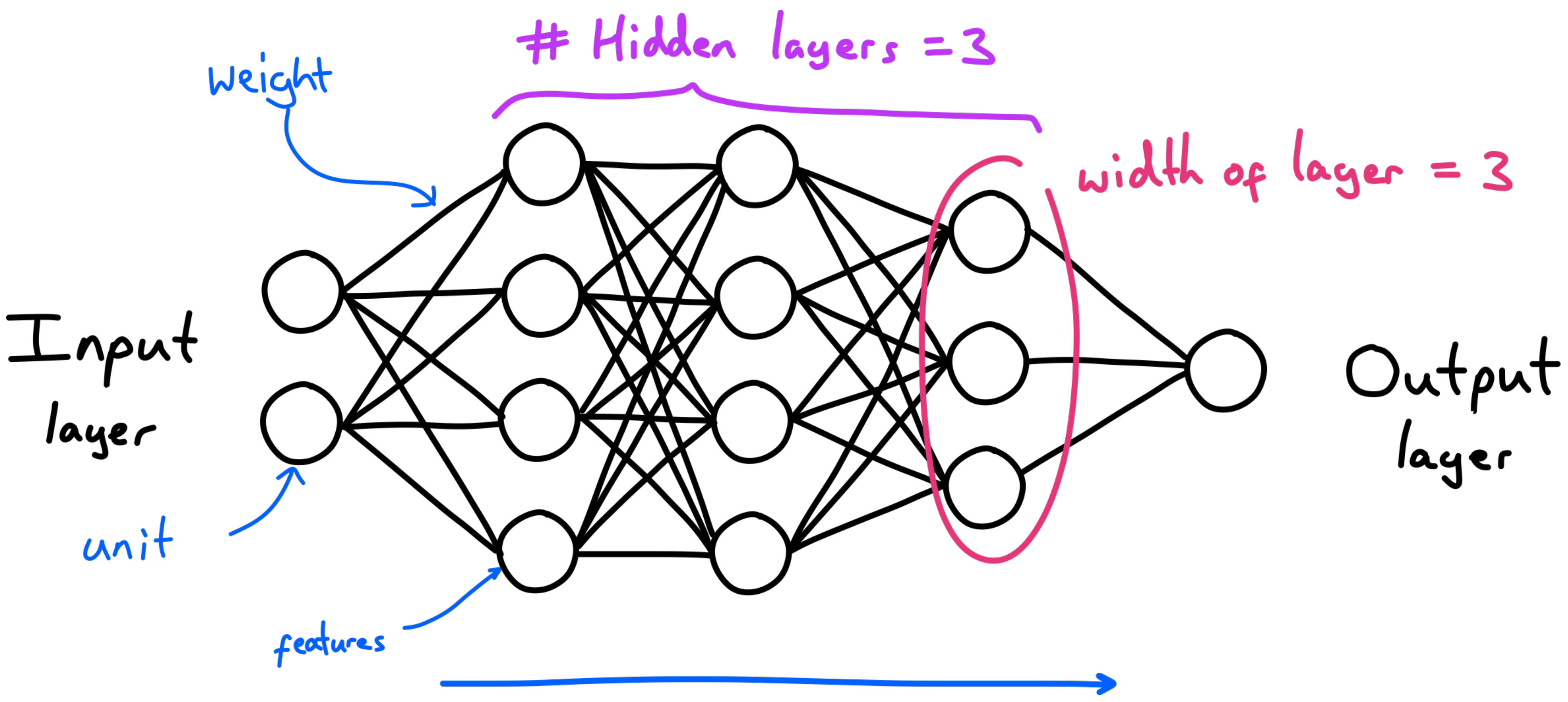
Recurrent neural networks

Neural network recap

Vanilla neural networks map one input to one output



Transformation $\hat{y} = w^{(4)} g(w^{(3)} g(w^{(2)} g(w^{(1)} x + b^{(1)}) + b^{(2)}) + b^{(3)}) + b^{(4)}$

They do this by building features from the input data, and then repeatedly building more complex features from those

We train these models by adjusting their parameters in a way that would reduce the instantaneous error

$$W \leftarrow W - \alpha \frac{\partial L}{\partial W}$$

Limitations of feedforward neural networks

What if we need outputs of a varying size?

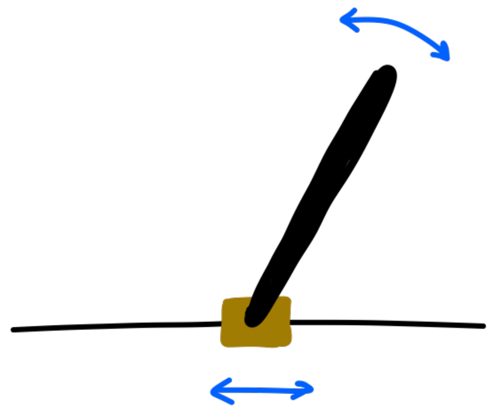
Text responses

What if our data is sequential?

Stock prices

What is a recurrent model?

Consider a dynamic system, where what happens next depends on what is happening now and what has happened previously



This is a drawing of a cart trying to balance the pole by moving side to side

Physics determines how the state will change over time

How the cart should move obviously depends on the pole's current position...

...but it also depends on its momentum

state at next timestep

what will happen next

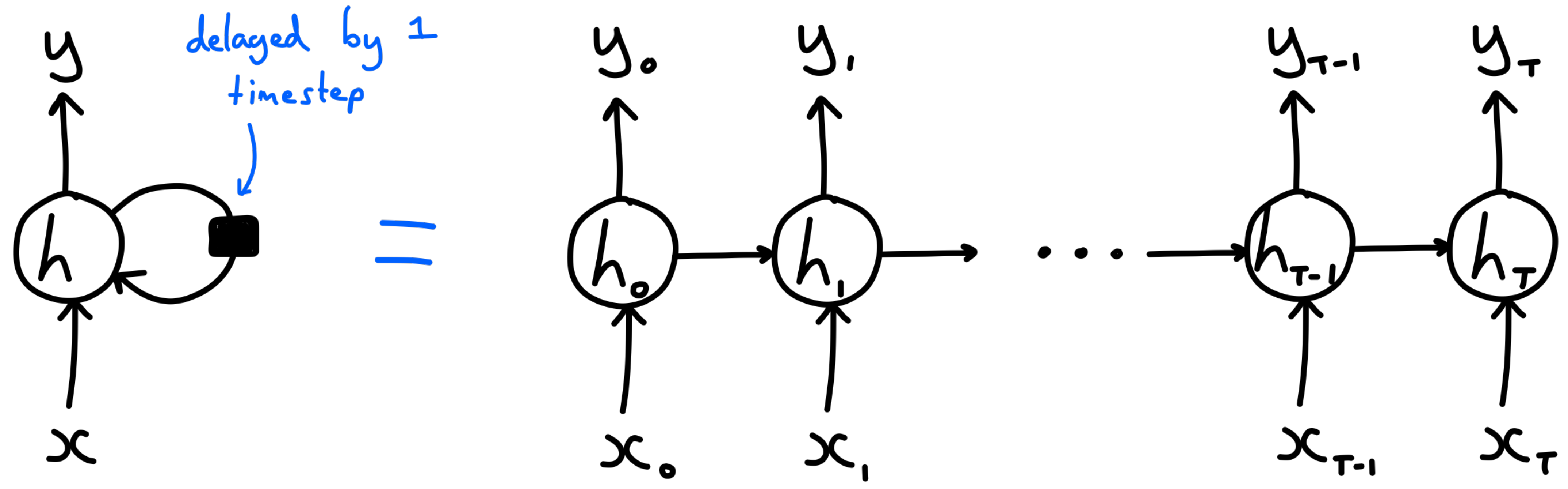
state at t ← summary of what has happened previously

input at t ← what is happening now

$$S_{t+1} = f(\underbrace{S_t \, , \, x_t}_{\text{how this will change what is happening}})$$

A recurrent model is influenced by is it's previous state, as well as by what is happening currently

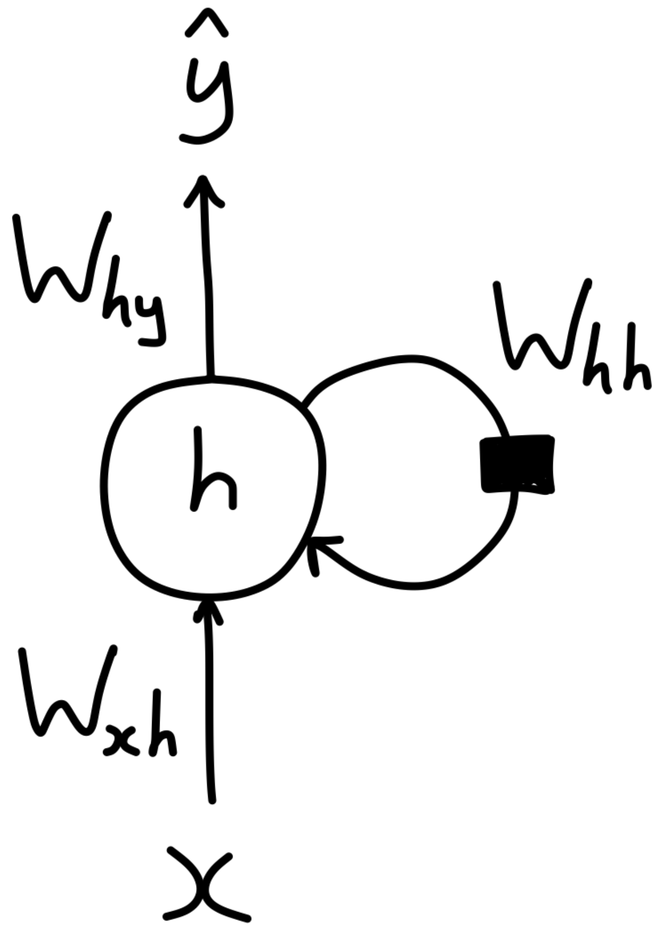
RNNs



Linear transformations (matrix multiplication plus bias) are applied between input-hidden, hidden-hidden and hidden-output.

An activation function is applied to the hidden state values.

Recurrent neural network representation



$x, h \in \mathbb{R}^h$ $W_{xh}, W_{hh} \in \mathbb{R}^{h \times h}$ $W_{hy} \in \mathbb{R}^{o \times h}$

vector state

$$h_{t+1} = W_{xh} x_t + W_{hh} h_t$$

$$y_t = W_{hy} h_t$$

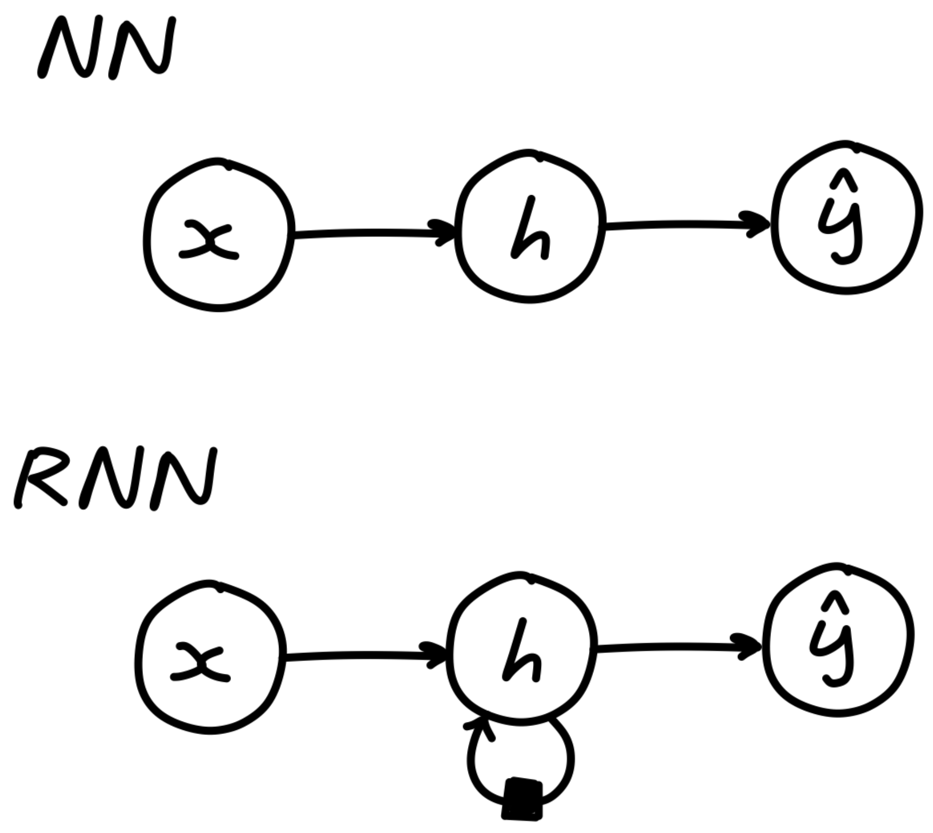
$$\underbrace{W = \begin{bmatrix} W_{hh} \\ W_{xh} \end{bmatrix}}_{\in \mathbb{R}^{2h \times h}} \qquad h_t = W^T \underbrace{\begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}}_{\in \mathbb{R}^{2h}}$$

Another view...

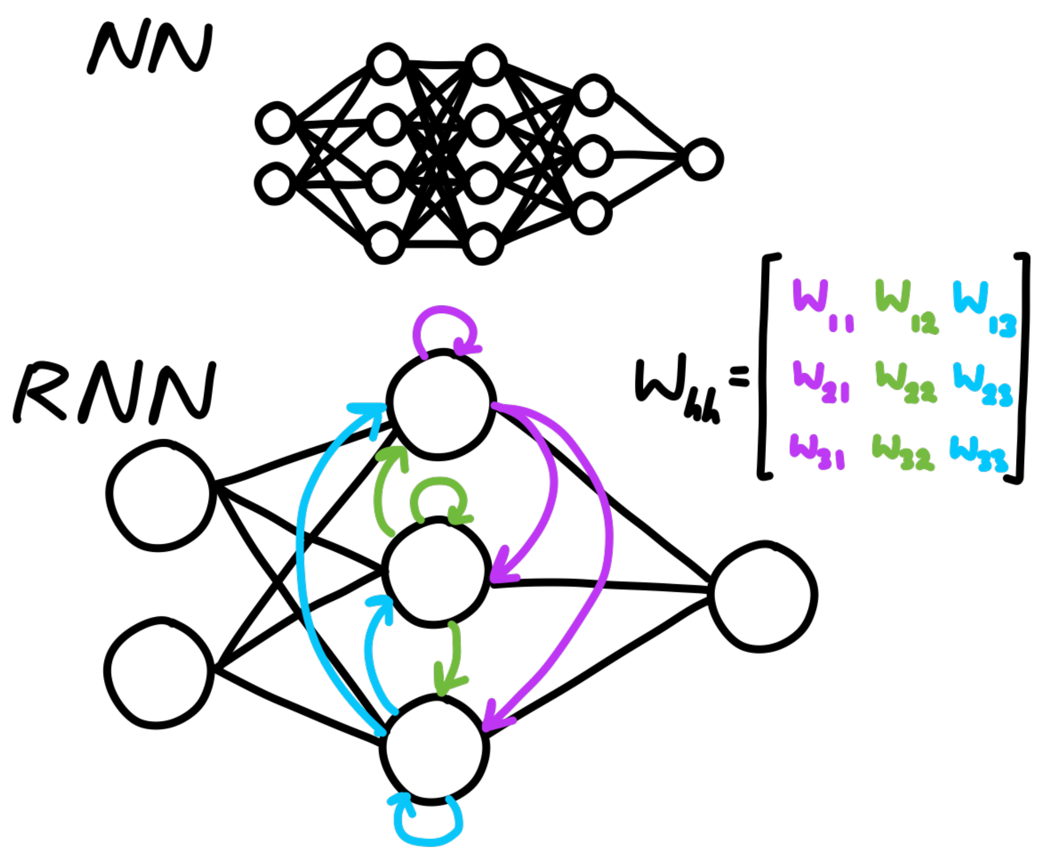
The RNN diagram is actually a more compact version of how we have drawn neural networks before.

Here, each node in the graph can be a vector of numbers, not just a scalar

Compact form



Full form


NN
RNN
$W_{hh} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix}$

You can see why we use the compact form...

Remember that the hidden, input and output values can be vectors, not just scalars

We can have many different types of RNNs

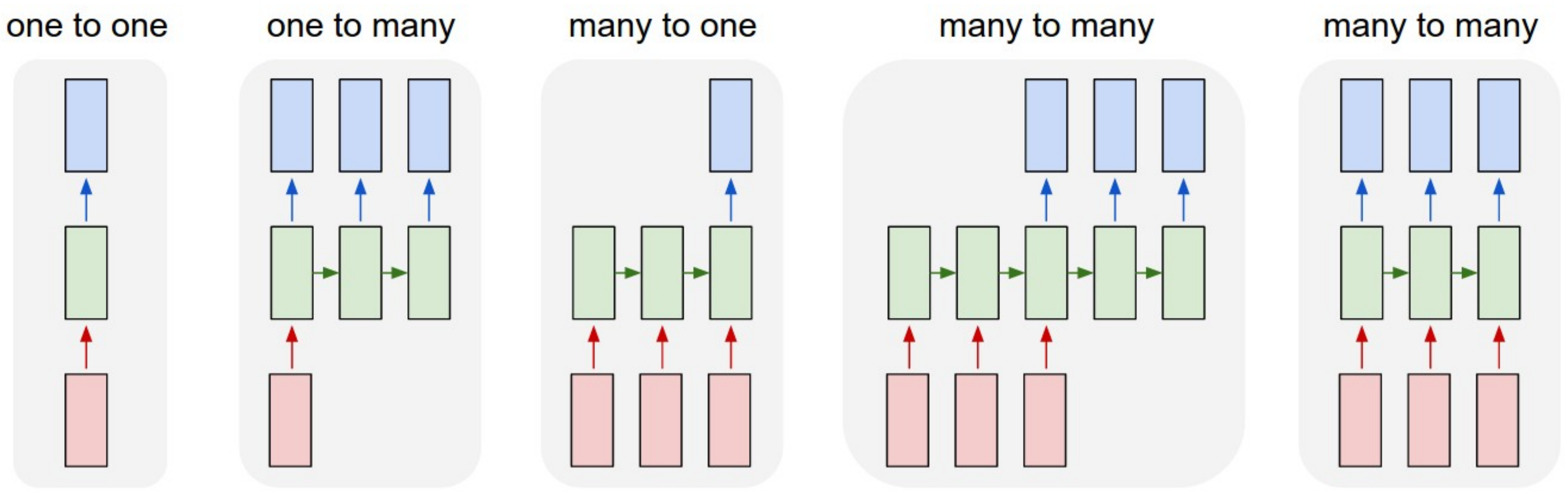


Image captioning

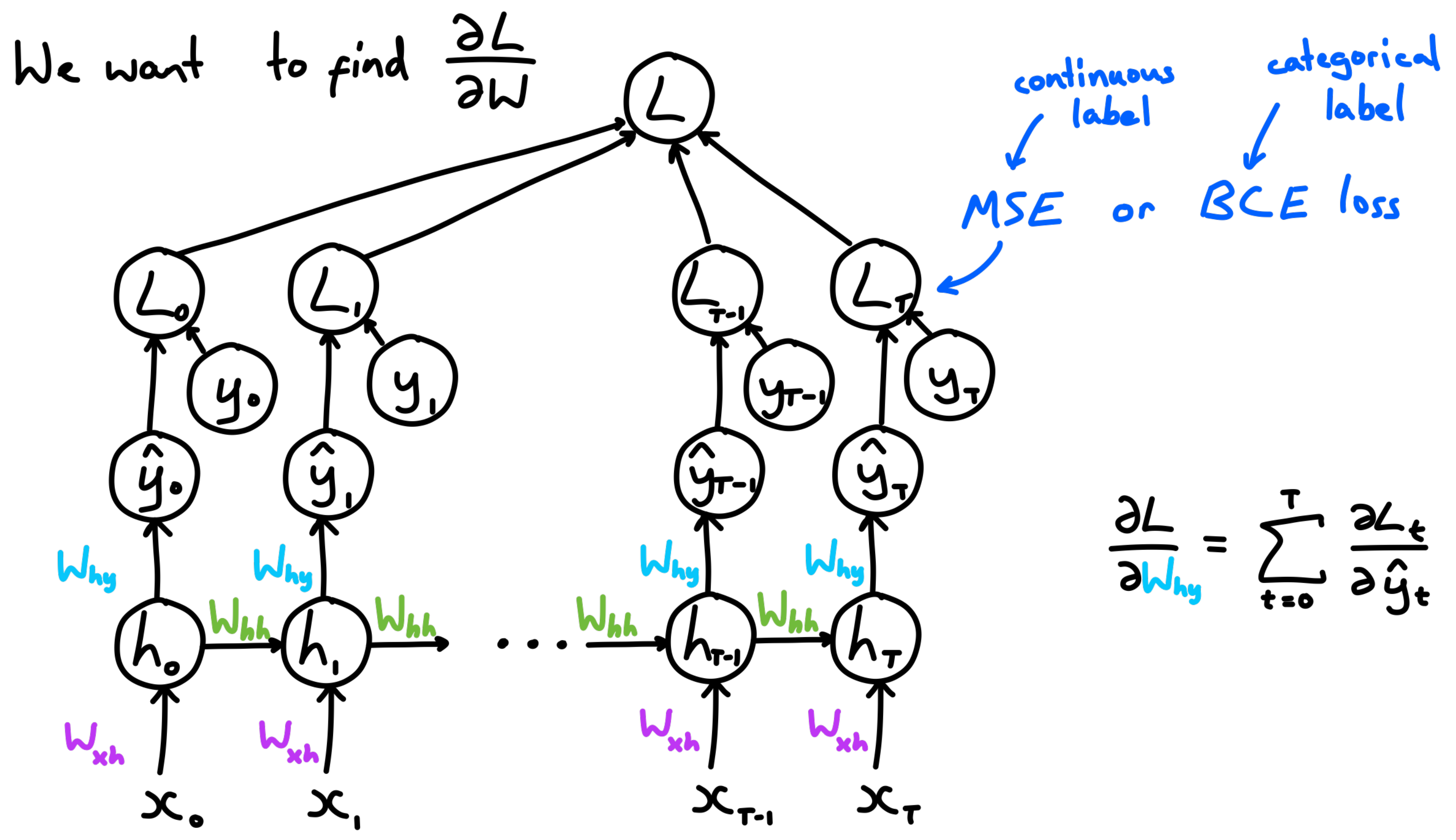
Sentiment analysis

Machine translation

Video frame classification

Training vanilla RNNs - backpropagation through time (BPTT)

We want to find $\frac{\partial L}{\partial W}$



$$\frac{\partial L}{\partial W_{hy}} = \sum_{t=0}^{T} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W_{hy}}$$

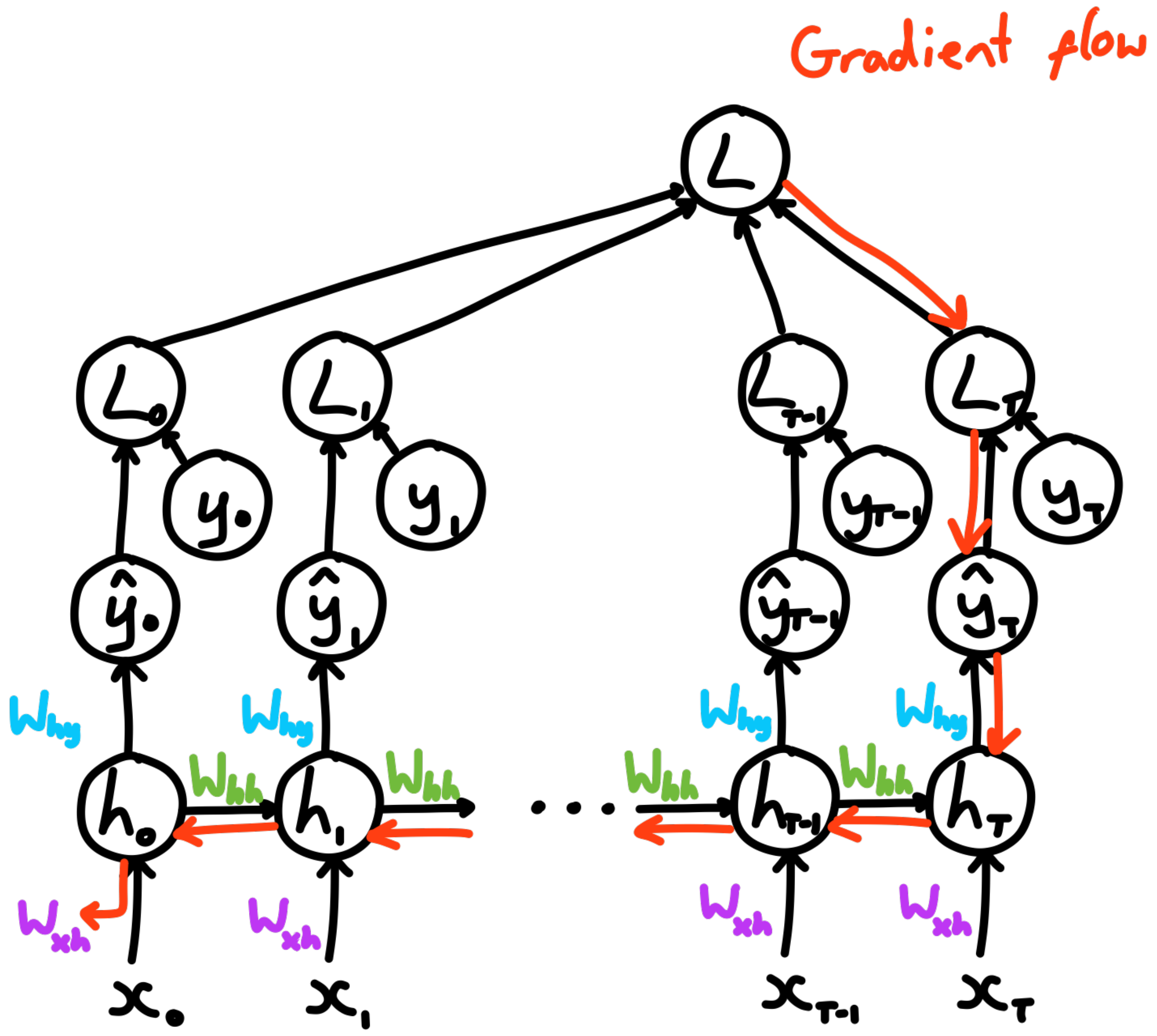
The loss (and hence it's derivative) is a function of the history of hidden states

We compute it in the same way as regular backpropagation, but sum up the gradients over time. Hence it is called backpropagation through time (BPTT).

Problems with this training

Unparralellisable computation

Because each step can only be computed after the previous one, the gradient computation time for one example varies linearly with the length of each sequence



$$L = \sum_{t=0}^{T} L_t(\hat{y}_t, y_t)$$

$$\begin{aligned}\hat{y}_t &= W_{yh} h_t \\ &= W_{yh}(W_{hh} h_{t-1} + W_{xh} x_t) \\ &= W_{yh}(W_{hh}(W_{hh} \underbrace{h_{t-2}}_{f(h_{t-3}, W_{hh}, W_{xh})} + W_{xh} x_{t-1}) + W_{xh} x_t)\end{aligned}$$

These very powerful network are hence very expensive to train

Problems with this training

Vanishing/exploding gradient

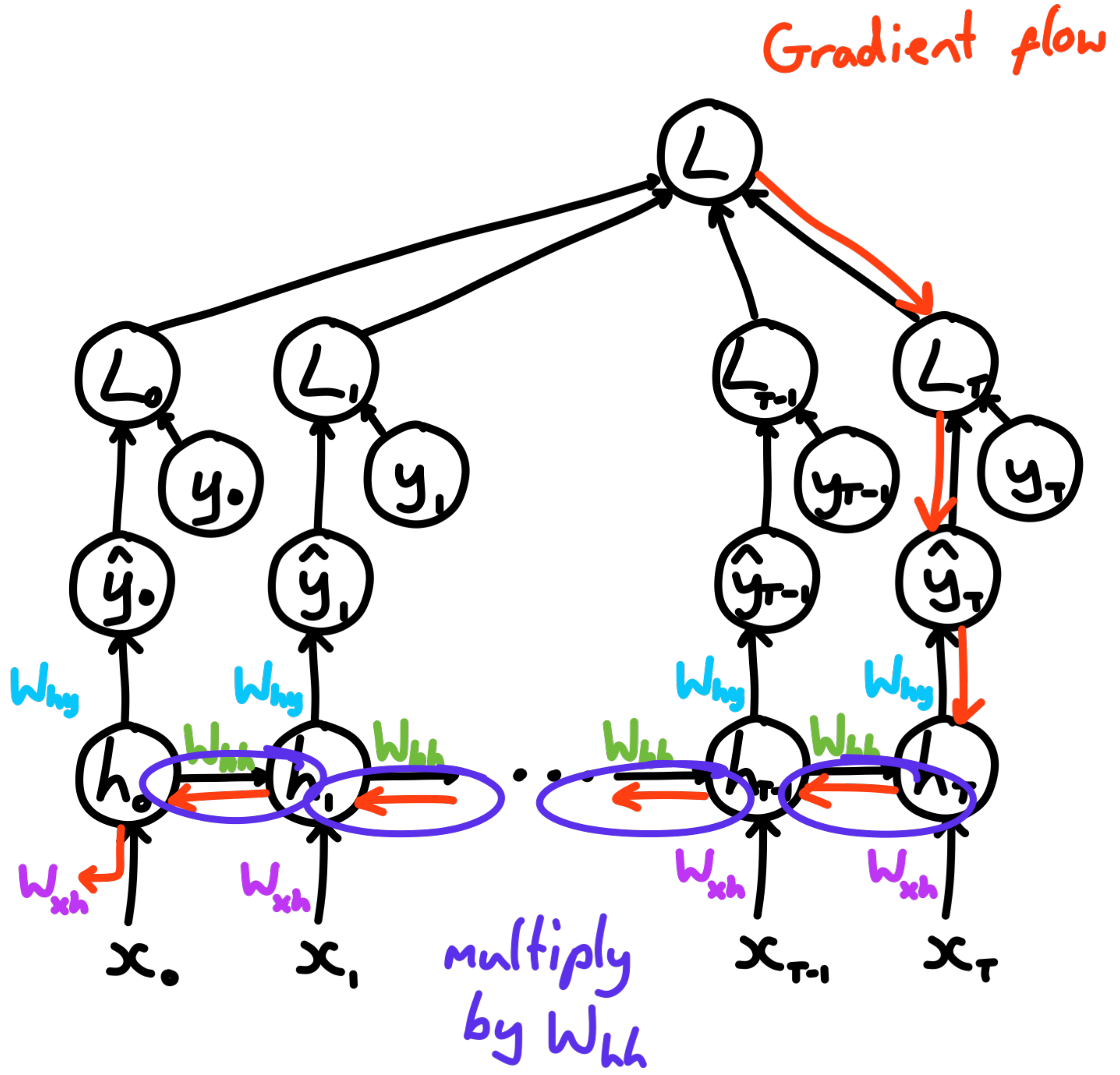
During computation of the gradient, the hidden-hidden weight matrix is applied repeatedly over time

This stretches or squashes the values by the same amount over and over again

Over many timestamps, this can either shrink the gradient to zero or explode it exponentially

The exploding gradient can be tackled by clipping it at a maximum value.

The vanishing gradient problem can be tackled using a different architecture... the LSTM

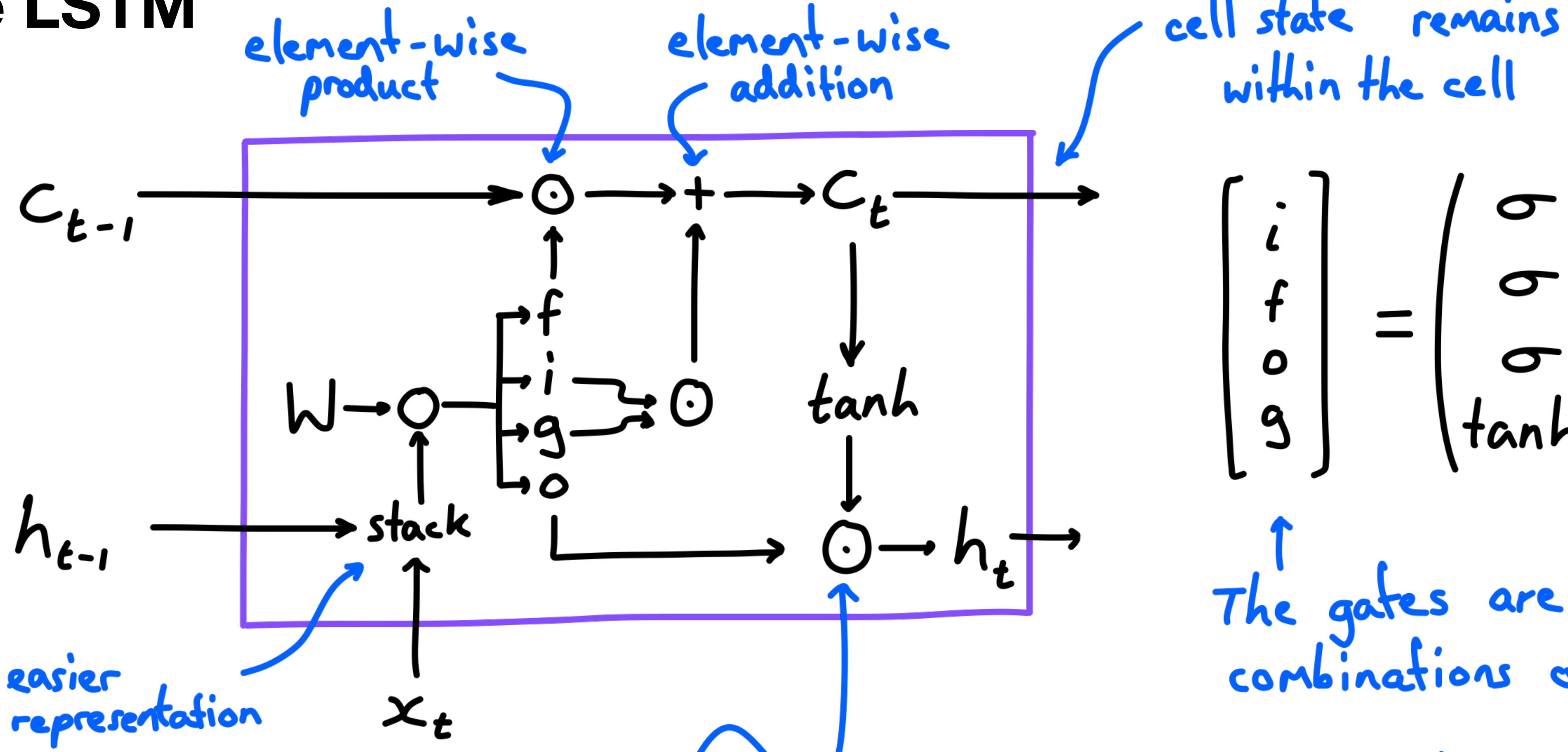


$$\hat{y}_t = W_{yh}\left(W_{hh} h_{t-1} + W_{xh} x_t\right)$$

$$= W_{yh}\left(W_{hh}\left(W_{hh} \underline{h_{t-2}} + W_{xh} x_{t-1}\right) + W_{xh} x_t\right)$$

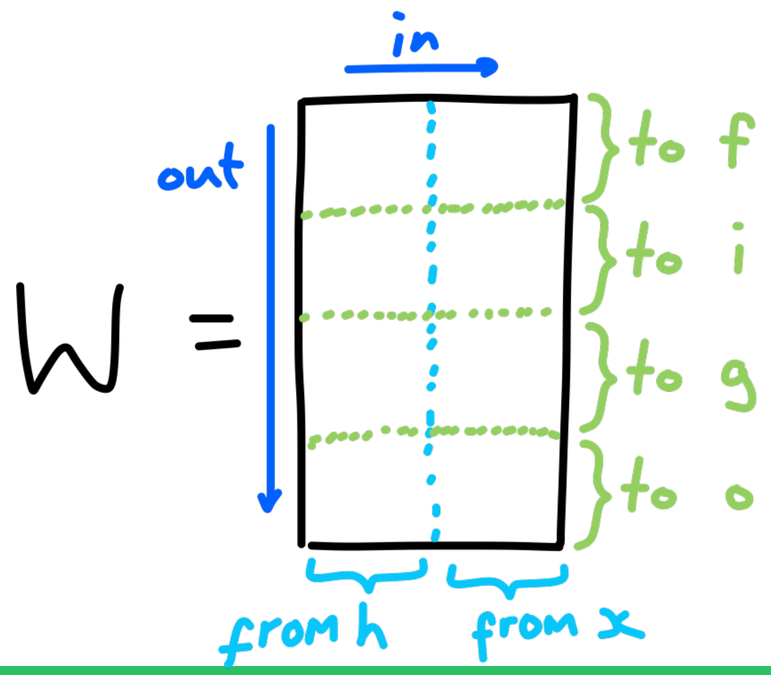
$\hookrightarrow f(W_{hh})$ and so on...

The LSTM



$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

The gates are activated linear combinations of h & x



out gate determines how much of the cell state to show in the next hidden state

how much of c to forget

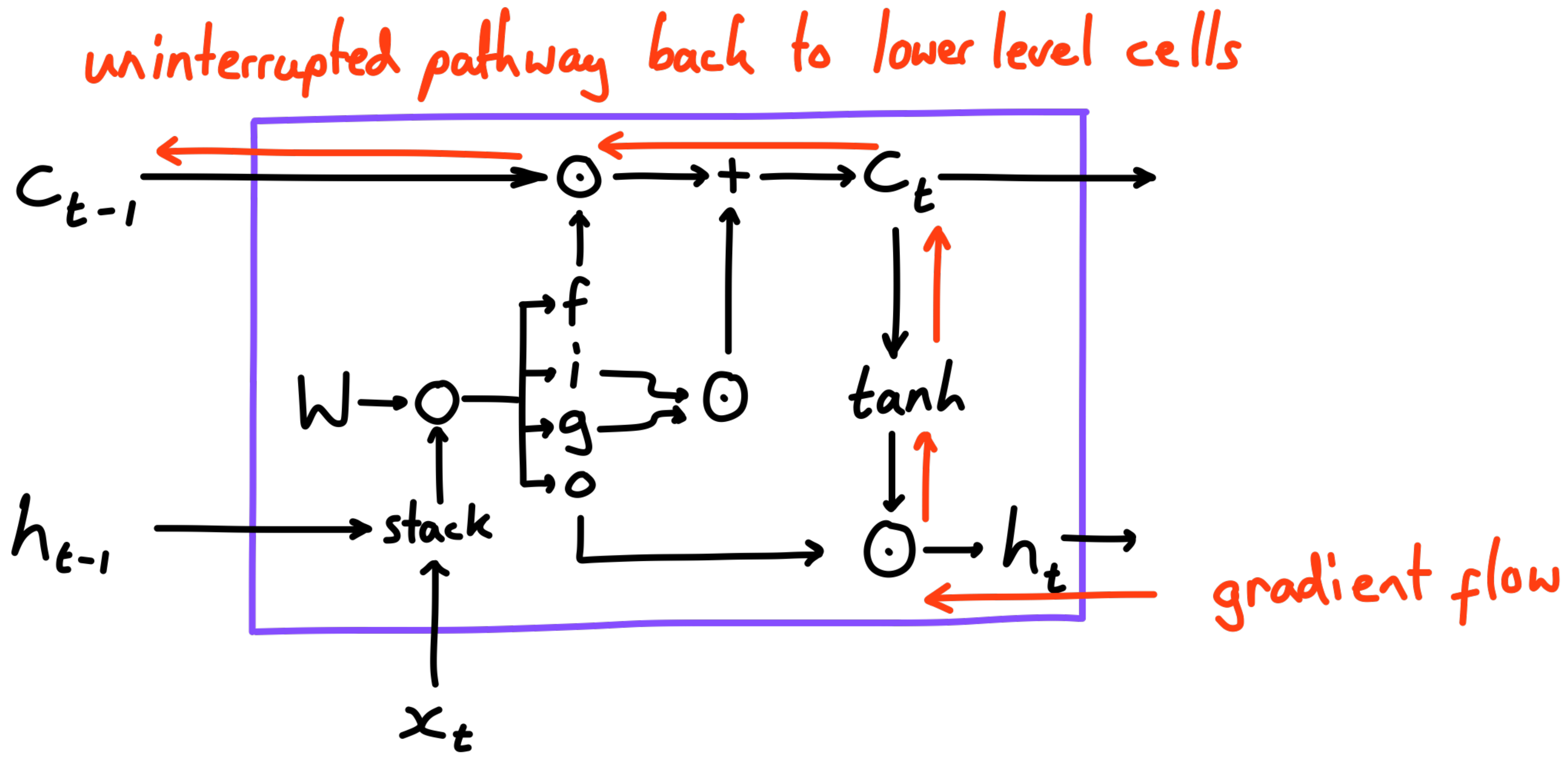
how much to input into next cell state

$$c_t = f \odot c_{t-1} + i \odot g$$

increment or decrement cell state values

$$h_t = o \odot \tanh(c_t)$$

How does this help gradient flow?



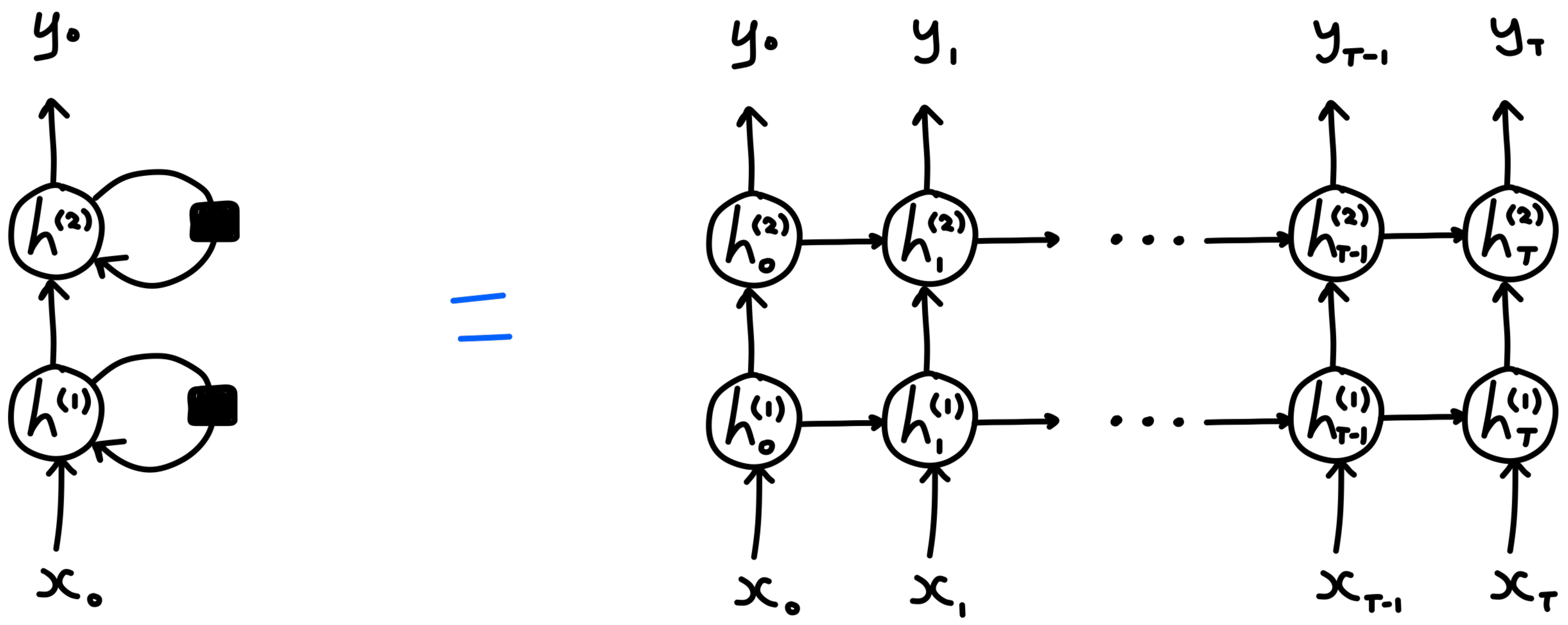
There is no repeated application of a matrix to the gradient

The multiplication is element wise

And the multipliers (elements of the cell state) change every timestep

Going deep

Instead of predicting an output through a single recurrent layer, we can pass those output features to another recurrent layer and build a deep recurrent model



y_0
$h^{(2)}$
$h^{(1)}$
x_0
=
y_0 y_1 y_{T-1} y_T
$h^{(2)}_0$ $h^{(2)}_1$ $h^{(2)}_{T-1}$ $h^{(2)}_T$
$h^{(1)}_0$ $h^{(1)}_1$ $h^{(1)}_{T-1}$ $h^{(1)}_T$
x_0 x_1 x_{T-1} x_T