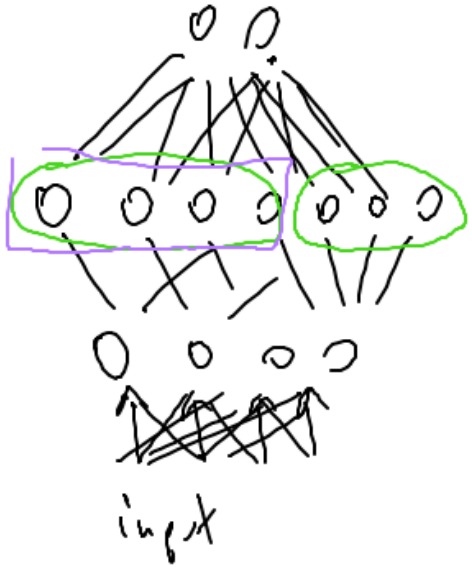
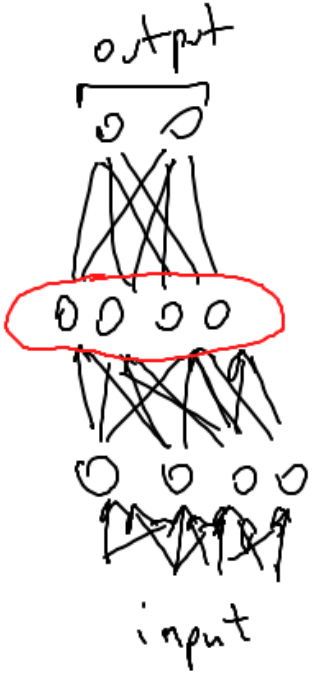
Dropout





or maybe ?

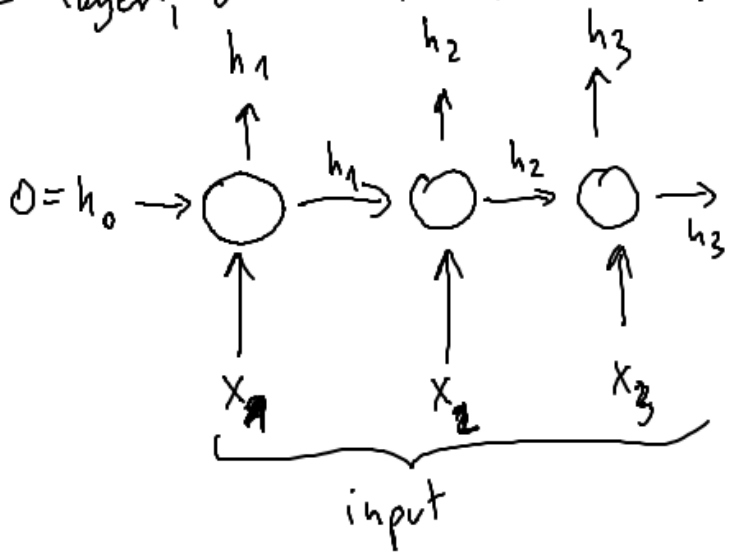
Let us say that we temporarily turn off the 4 neurons in [ ].
Then during the training we need to compensate that fact, and multiply the output from the remaining 3 neurons by $\frac{7}{3}$.
At the end we will have the sum of outputs of all 7 neurons,
this sum = (the sum of 4) + (the sum of 3) = $\frac{7}{4}$( ) · $\frac{4}{7}$ + $\frac{7}{3}$·( ) · $\frac{3}{7}$

In the dropout, we randomly select some of the neurons in a layer that are going to be turned off (for that particular batch)

Usually one turns off 25% - 50%.

Recurrent Neural Networks (RNN)

one layer, one cell, input = sequence of length 3



$h_t \in \mathbb{R}$

$\in \mathbb{R}$

Simple RNN cell: $h_t = \sigma\left(W \cdot x_t + U \cdot h_{t-1} + b\right)$

trainable parameters (weights)

inputs to the cell

$W, U, b \in \mathbb{R}$

e.g. $h_1 = \sigma\left(W \cdot x_1 + U \cdot h_0 + b\right)$

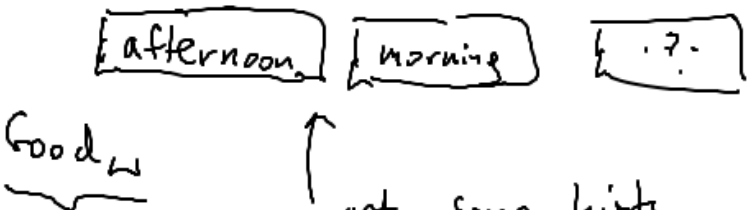
$h_2 = \sigma\left(W \cdot x_2 + U \cdot h_1 + b\right)$

$h_3 = \sigma\left(W \cdot x_3 + U \cdot h_2 + b\right)$

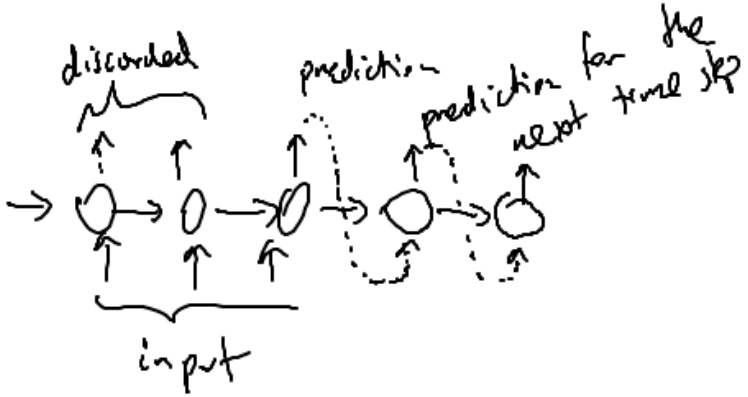
One can treat h_3 as a prediction for the fourth element of the sequence (x_1, x_2, x_3)

Some applications:

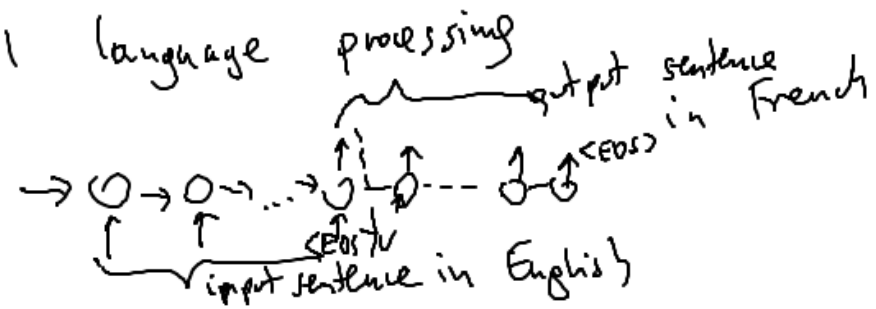
- Texting on a smartphone



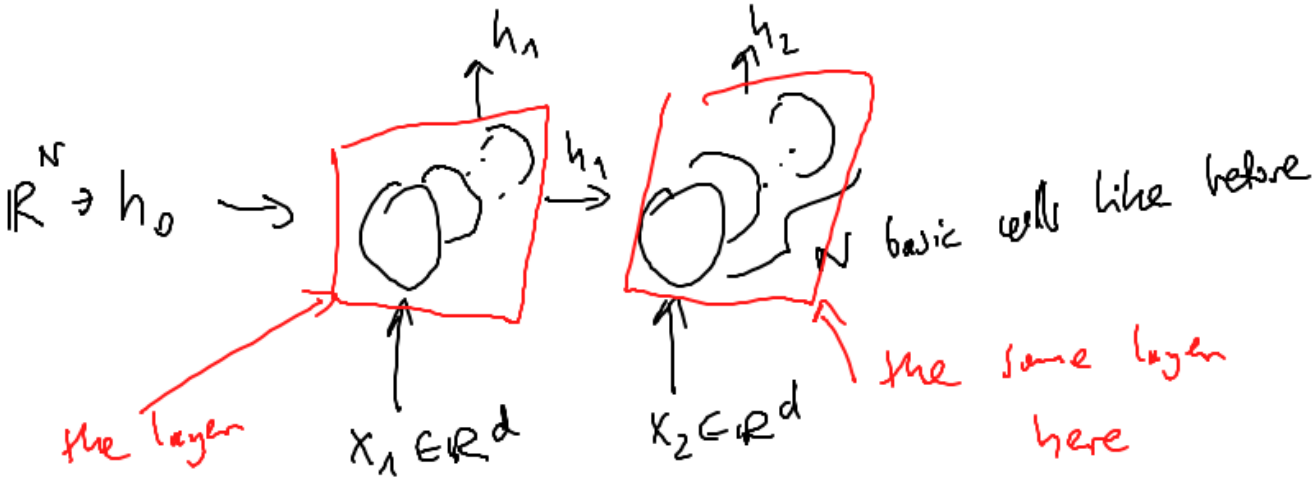
- Predicting some time series



* In natural language processing



Generalisation to a multidimensional case with multiple neurons in a layer



simple RNN

← one layer

← picture with unrolling

A picture without unrolling:

$$h_t = \sigma\left(W \cdot x_t + U \cdot h_{t-1} + b\right)$$

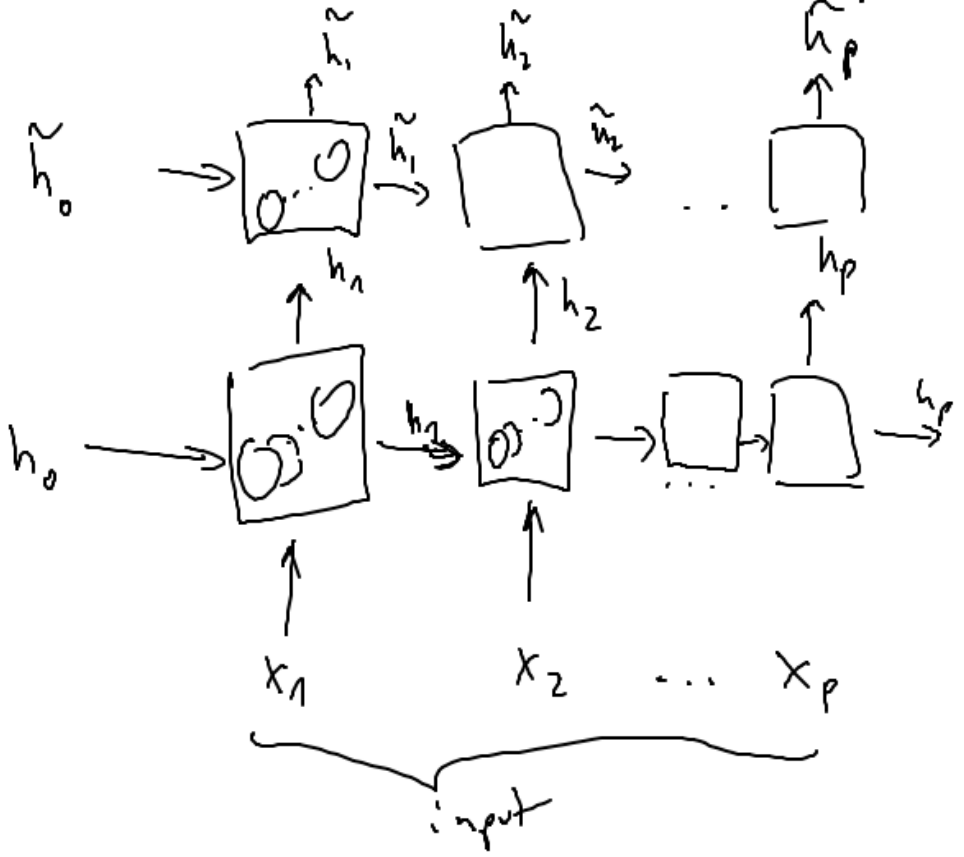
$x_t \in \mathbb{R}^d$

matrix $N \times d$ · $[\,]$ $d \times 1$ · matrix $N \times N$ · $\in \mathbb{R}^N$ · $[\,]$ $N \times 1$ · $\in \mathbb{R}^N$

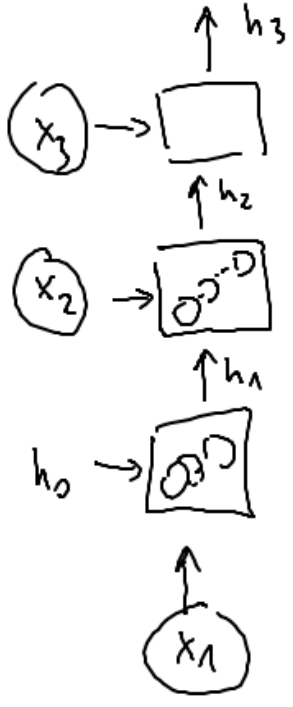
$\in \mathbb{R}^N$

$$= \begin{bmatrix} \sigma(\dots) \\ \sigma(\dots) \\ \vdots \\ \sigma(\dots) \end{bmatrix} \in \mathbb{R}^N$$

One can have multiple layers: ... e.g. dense layer



Other way of looking at a single RNN layer:



When a sequence has 3 elements, then the RNN layer is similar to a classical NN with 3 layers.

Usual problems with vanishing / exploding gradients.

LSTM (Long - Short Term Memory)

95 - 97 - 99
Hochreiter, Schmidhuber

$$f_t = \sigma_g\left(\underbrace{W_f}_{N\times d}\cdot x_t + \underbrace{U_f}_{N\times N}\cdot \underbrace{h_{t-1}}_{\in\mathbb{R}^N} + \underbrace{b_f}_{\in\mathbb{R}^N}\right)$$ forget gate

$$i_t = \sigma_g\left(W_i\cdot x_t + U_i\cdot h_{t-1} + b_i\right)$$ input gate

$$o_t = \sigma_g\left(W_o\, x_t + U_o\, h_{t-1} + b_o\right)$$ output gate

$$\tilde{c}_t = \sigma_c\left(W_c\, x_t + U_c\, h_{t-1} + b_c\right)$$

Standard choice:
$\sigma_g(x) = 1/(1+e^{-x})$
$\sigma_c(x) = \sigma_h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

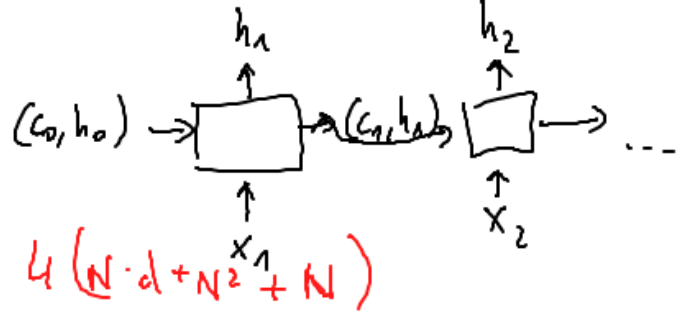
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$ ← the memory cell

$$h_t = o_t \cdot \sigma_h(c_t)$$ ← the output and the hidden state

$\odot$ = element-wise product

$\{\ x_t \in \mathbb{R}^d,\ f_t, i_t, o_t, h_t \in \mathbb{R}^N$

$d = N = 1 \longrightarrow$ 12 weights



$4\left(N\cdot d + N^2 + N\right)$