CNN:

Input Tensor → [#ex, #ftr] → X

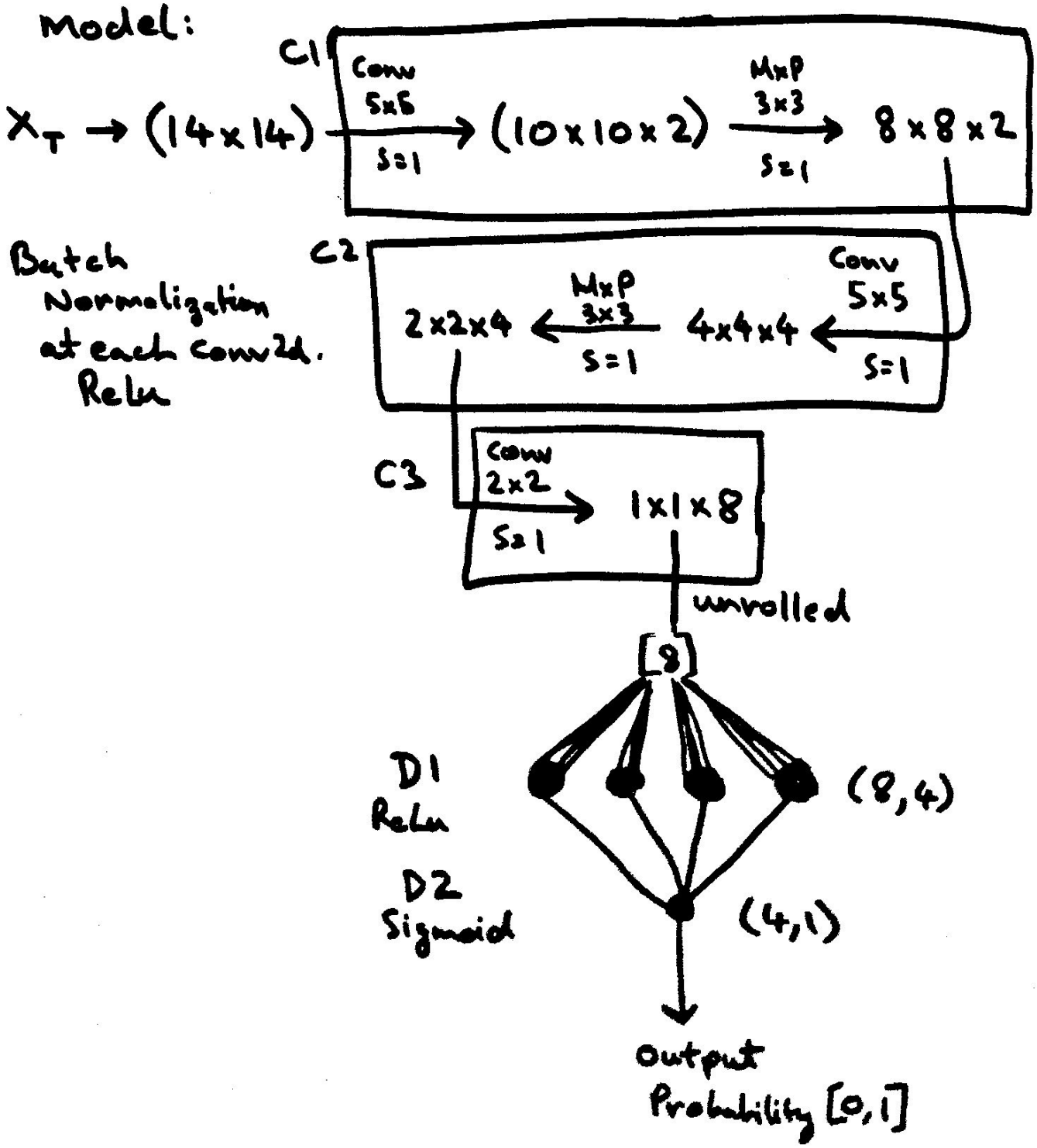
① Transform to set of 2D images with 1 channel.

↳ [#ex, #ftr] $\xrightarrow{T}$ [#ex, 1, $\sqrt{ftr}$, $\sqrt{ftr}$]

↳ X_T

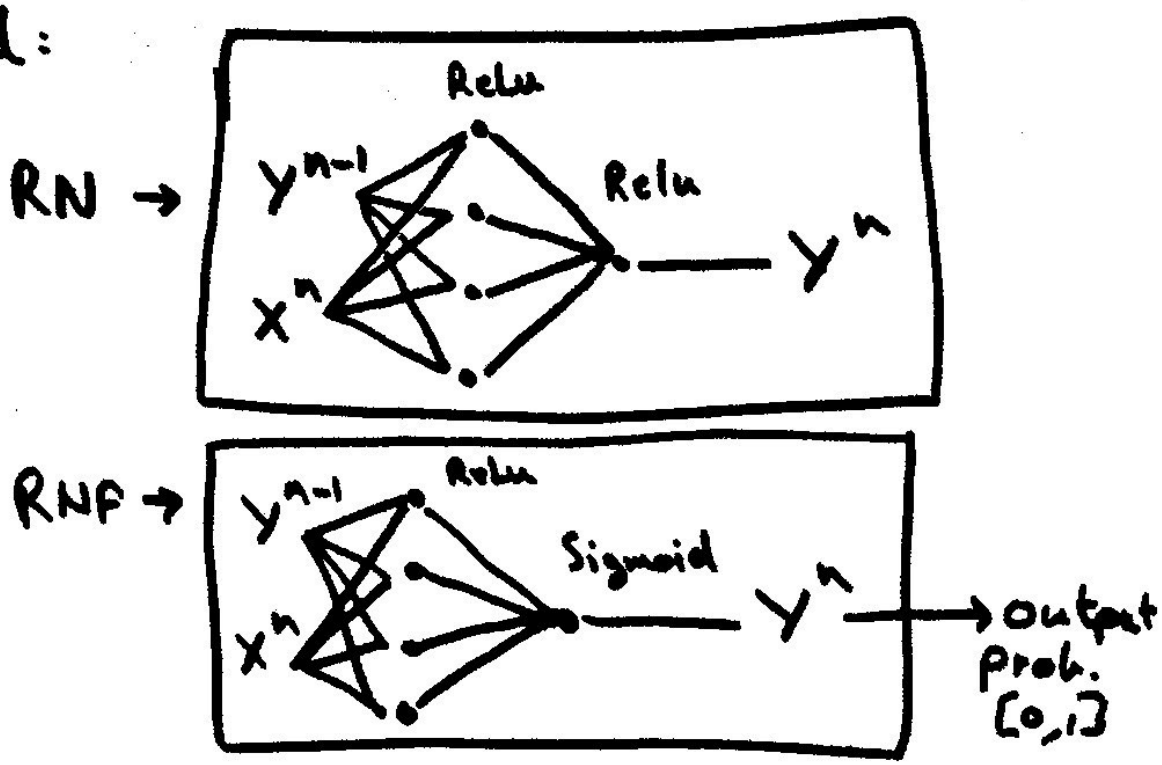
② Feed to CNN & train.

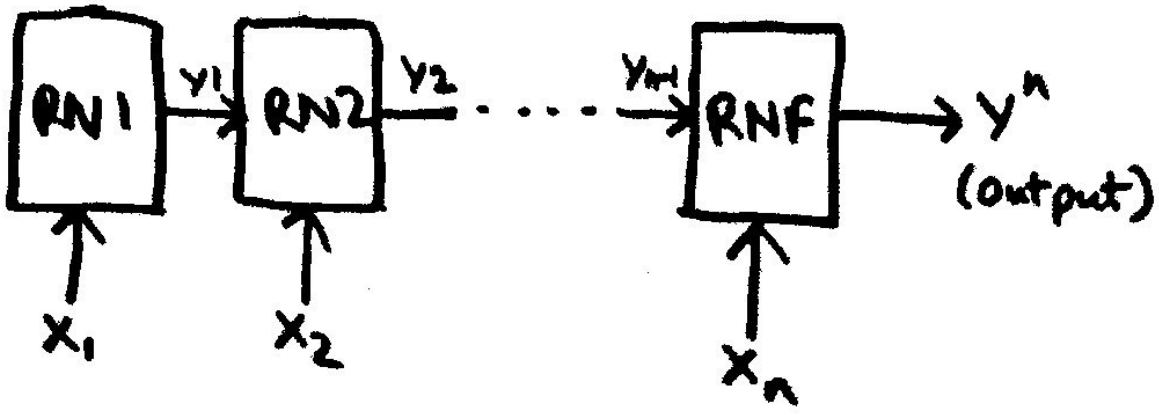
Model:



RNN: $X \rightarrow [x_1, \ldots x_n] \rightarrow$ features.

model:





MLR →

$$X = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \longrightarrow f(x) \rightarrow \text{Prediction Probability } [0, 1]$$

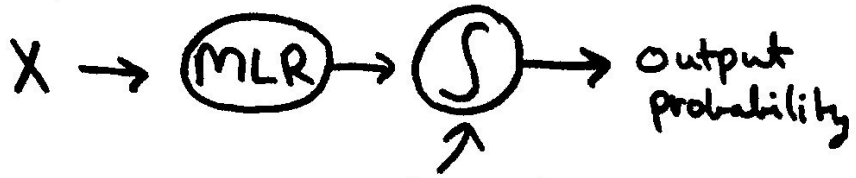
$$f(x) = \omega_0 x_0 + \omega_1 x_1 + \ldots\ldots \omega_n x_n + b$$

here, $\{\omega_0, \ldots \omega_n\}$ and b are trainable parameters.

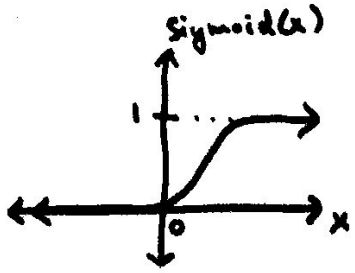
$\omega_0, \ldots \omega_n$ ↑ weights

b ↑ bias

Perceptron →



ex- Sigmoid $(x) = \frac{1}{1+e^{-x}}$ →

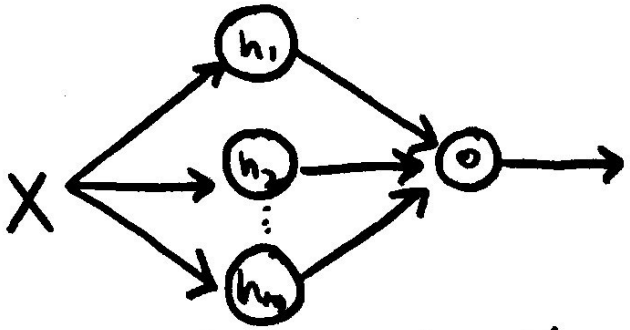


So, expanding, (lets call sigmoid(x) as $g(x)$),

$$\text{output prob} \rightarrow Y = g\left(\sum_n \omega_n x_n + b\right)$$

where ω_n & b are trainable params.

Shallow NN →

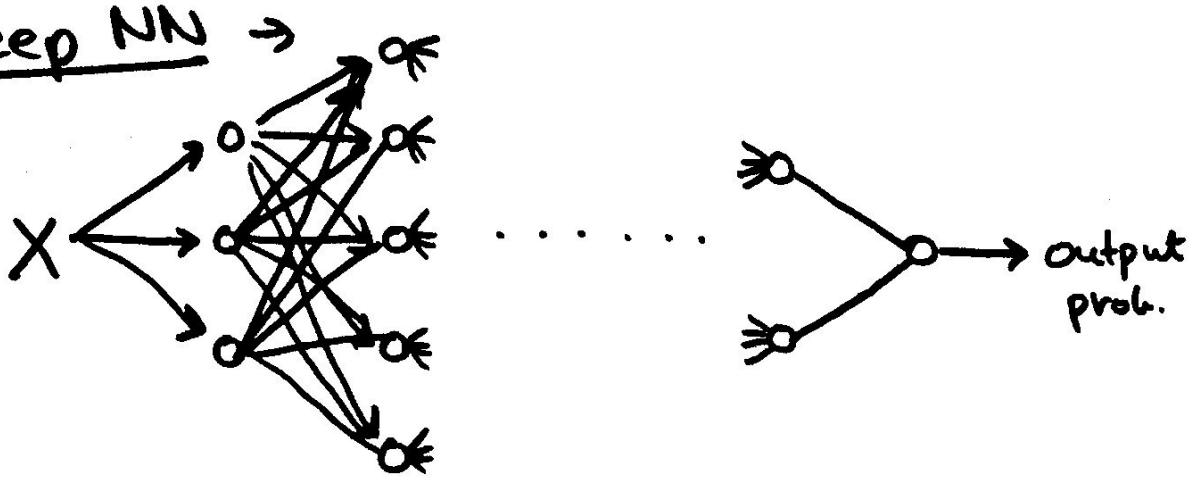


output probab. $= Y = g_o\left(\sum_m w_m g_m\left(\sum_n x_n + b_h\right) + b_o\right)$

Shallow NN introduces a hidden layer that comprises of "neurons"/nodes that themselves structurally the same as a Perceptron but usually have a different non linearity than Sigmoid. In our case, its ReLU non-lin.

The idea is that the individual nodes can "learn" to "specialize" in "seeing" different patterns & thus in theory improving performance.

Deep NN →



Same as shallow but with more than 1 hid. layers.
Idea is that lower layers extract low level info & higher up you go, more complex information is being processed by the 'neurons'.

Deep NN can significant by improve a model's ability to extrapolate otherwise abstract patterns.

General rule of thumb, as $L \uparrow$, # nodes per layer can $\downarrow$ since individual neuron specialization is not required. Thus we are able to extract more abstract information.

$$\text{output } Y = g_o\left(\sum \omega_i^o g_{o-1}^i(\cdots) + b_o\right)$$