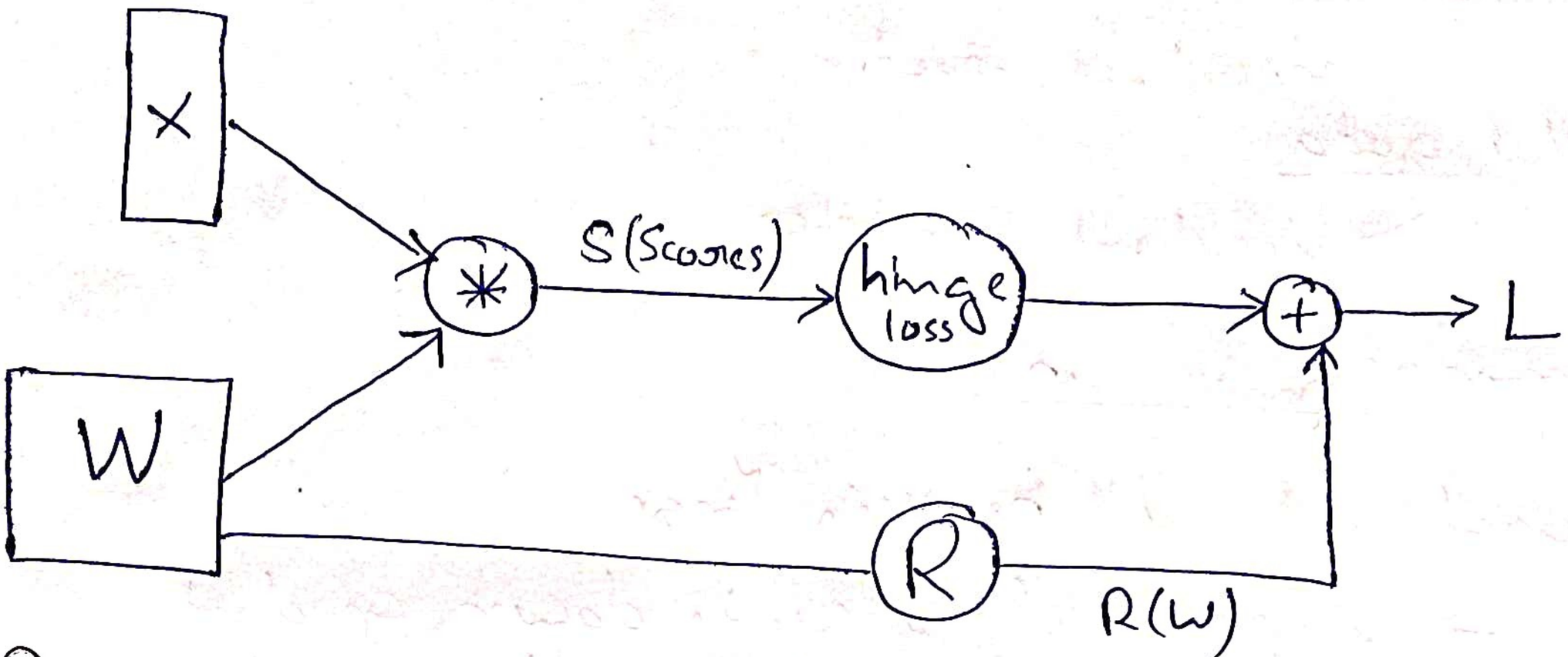
(4) Backpropagation & Neural Network

★ Computational Graphs

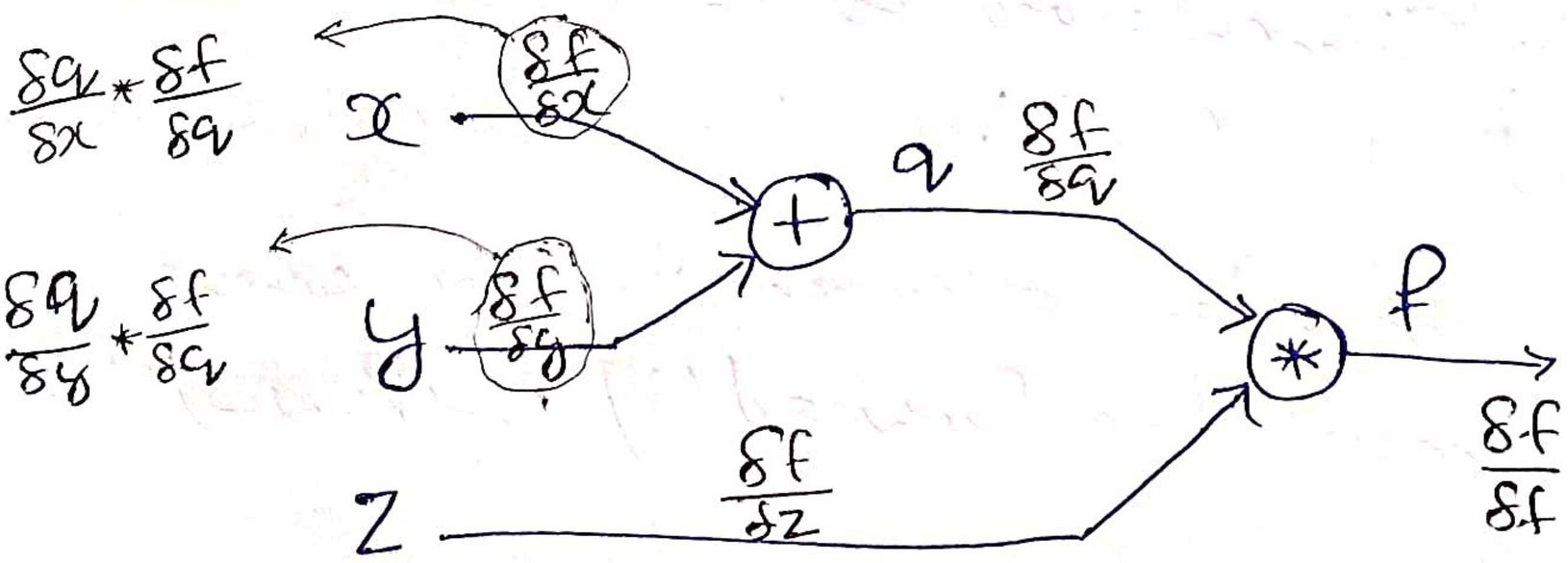
⇒ A Computational graph is defined as a directed graph where the nodes correspond to mathematical operations.



★ Backpropagation

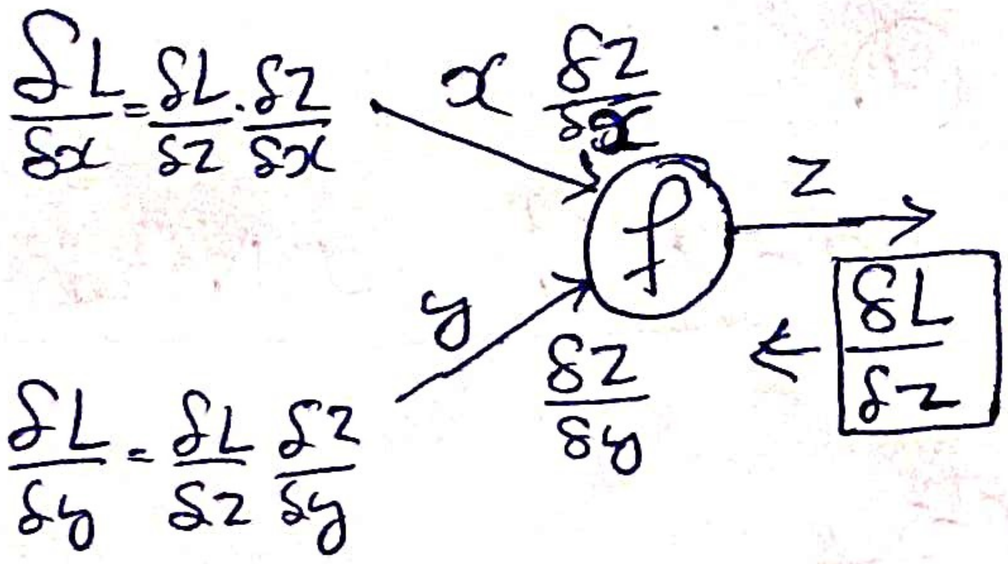
Example

$$f(x,y,z) = (x+y)*z$$



$$q = x+y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q$$



★ Patterns in backward flow

* **Add gate**
  → Gradient distributer
* **Max gate**
  → Gradient routes
* **Mul gate**
  → Gradient switcher

★ Summary so far

⇒ Neural nets will be very large
  ↳ Impractical to write down gradient formula by hand for all parameters

⇒ **Backpropagation**
  ↳ Recursive application of the chain rule along a computational graph to compute the gradients of all input/parameters/intermediates.

⇒ Implementations maintain a graph structure, where the nodes implement the forward() / backward() API.

forward
  ↳ Compute result of an operation & save any intermediates needed for gradient computation in memory.

backward

→ Apply the chain rule to compute the gradient of the loss function with respect to the inputs.

★ Neural Networks

⇒ Simple function stacked on top of each other, in hierarchical way, in order to make a more complex non linear function.

(Before) Linear Score function: $f = Wx$

(Now) 2-layer Neural Network: $f = W_2 \max(0, W_1 x)$

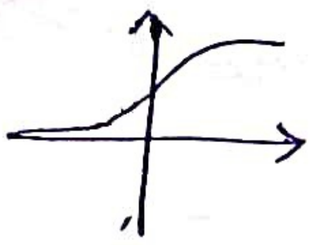
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

★ Activation Functions

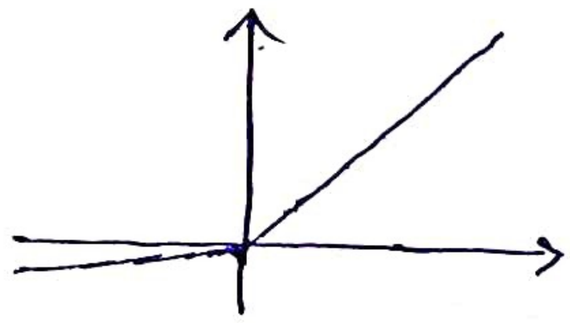
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



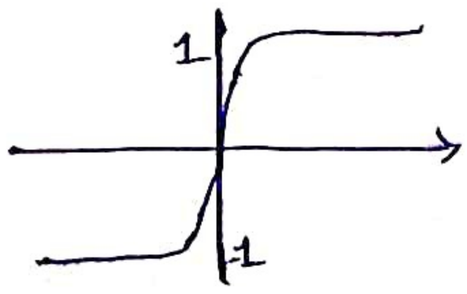
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

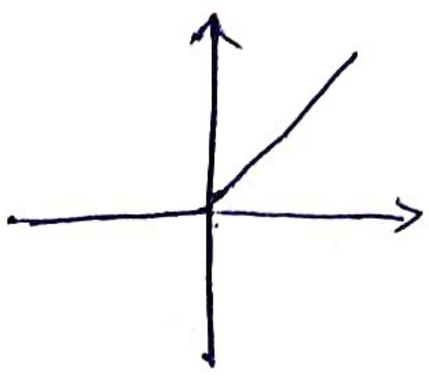


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

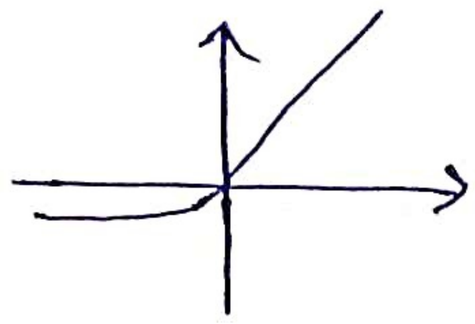
ReLU

$\max(0, x)$

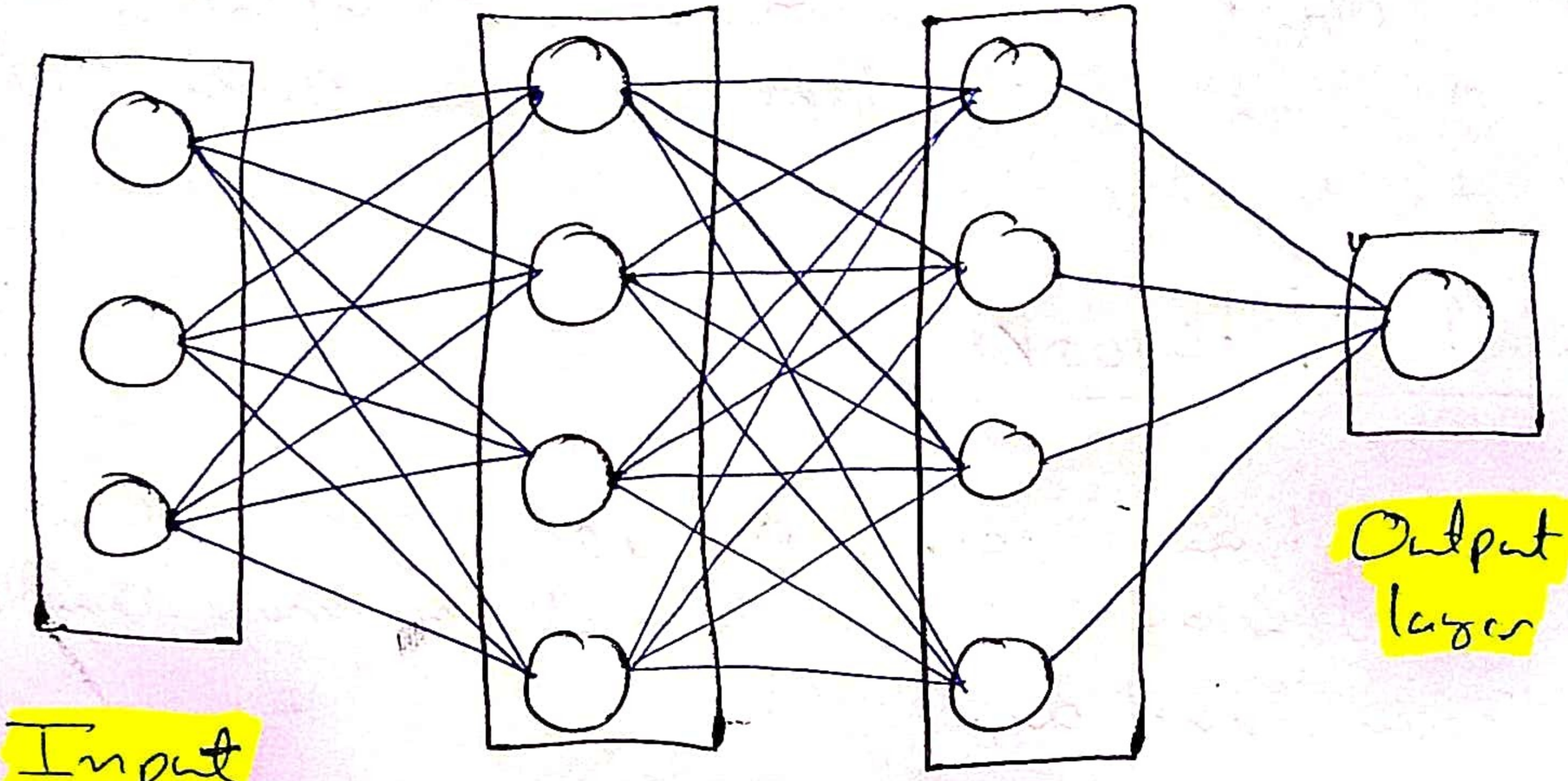


ELU

$$\begin{cases} x & a \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



★ Neural Networks : Architectures



Input layer

Output layer

{← Hidden layers →}

⇒ Above Exaple Shows "fully-Connected layers"