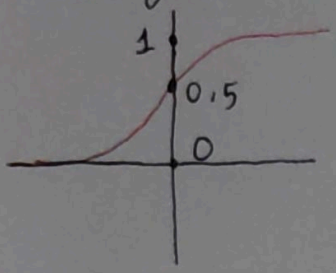
☆ Sigmoid

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

Sigmoid Activation = Logistic Activation



Any input is scaled to a value b/w 0-1

Eg $x=2 \Rightarrow f(x) = \frac{1}{1+e^{-2}} = 0.88080$

$x=-1 \Rightarrow f(x) = \frac{1}{1+e^{1}} = 0.26894$

$x=0 \Rightarrow f(x) = \frac{1}{1+1} = 0.5$

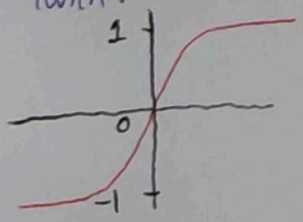
Advantages
- can act as a gate.
- Smooth gradient
  ⇒ "Jumps" in output values prevented
- Output value b/w 0 & 1
  ⇒ each neuron output normalized
- clear Predictions
  Very close to 1 or 0.

Disadvantages
- Sigmoid outputs are not zero-centered (efficiency of weight update effected)
- Power operations are relatively time consuming (slower for computers)
- Prone to gradient vanishing
  i.e. Vanishing Gradient Problem.

☆ Hyperbolic Tangent function (Tanh)

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

Tanh:

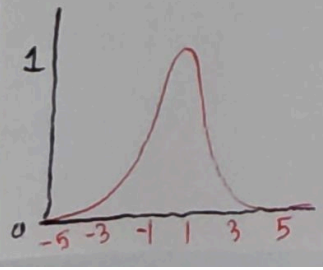


(i) Curves similar to sigmoid

(ii) whatever input (large or small), output is almost smooth & gradient is small, not great for weight update.

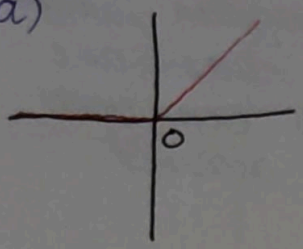
(iii) Better than Sigmoid
↳ ∵ output interval of tanh & whole func" is 0-centric.

Derivative of Tanh: $\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2$



Disadvantages
- Power operations are time consuming
- Saturated Tanh neuron causes the gradient to vanish
  ⇒ Vanishing Gradient issue

★ ReLU (Rectified Linear Unit)

(a) 

$R(z) = \max(0, z)$

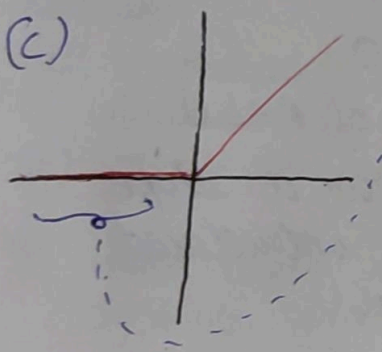
-ve input ⇒ output = 0
+ve input ⇒ Stays +ve
eg 7 → 7
8 → 8
-4 → 0

(b) derivative of ReLU

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \Rightarrow f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Problem → when input is -ve
⇒ output = 0
⇒ gradients will ~~do~~ die

Eg when initializing from the normal distribution $N(0,1)$, half of the values are -ve. Activating with ReLU means setting half of the values to 0.

(c) 

→ no "vanishing gradient", gradient in left already vanished.
⇒ This phenomenon is called "Dead-Neuron"

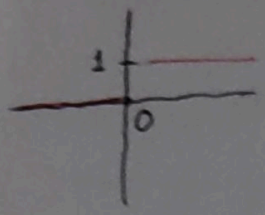
Dead-neuron: their output always 0 ∵ weighted sum of inputs ≤ 0

(d) → In theory no "very" good, but practically the work really work.
→ The right side "not vanishing" appears to be enough.
→ But time wasted & output is not 0-centric

(e) Advantages: (i) when input is +ve ⇒ no gradient saturation
(ii) Calculn faster than Tanh & Sigmoid (∵ they have exponents)

(f) fixing Dead Neurons → Leaky ReLU
→ P ReLU
→ Exponential ReLU

★ Problem of Dead neuron



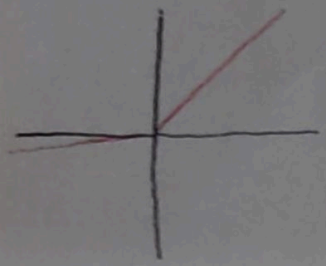
$$f'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

At -ve side of graph, gradient value = 0.
⇒ During backpropagation, weights & biases for some neurons are not updated. This can create dead neurons which never get activated.

★ Leaky ReLU

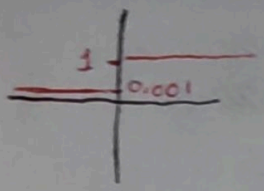
$$f(x) = \max(0.01x, x)$$

$$\Rightarrow f(x) = \begin{cases} x, & x \geq 0 \\ 0.01x, & x < 0 \end{cases}$$



(i) Solves dead neuron issue of ReLU

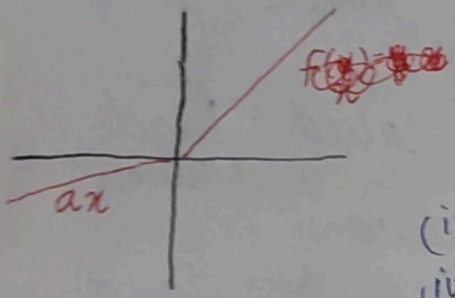
(ii) $$f'(x) = \begin{cases} 1, & x \geq 0 \\ 0.001, & x < 0 \end{cases}$$



Now gradient of left side of graph is non-zero
⇒ no longer dead neuron in that region.

★ Parameterised ReLU

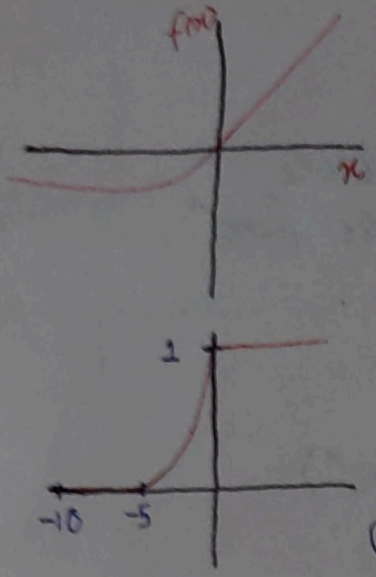
$$f(x) = \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}$$



(i) Solves dead neuron issue of ReLU
(ii) A new parameter ~~of slope~~ as a slope of the -ve part of func^n introduced
(iii) if $a = 0.01$ ⇒ Leaky ReLU
(iv) 'a' = trainable parameter
(v) Use when Leaky ReLU func^n still fails to solve the problem of dead neurons and the relevant information is not successfully passed to the next layer.

☆ Exponential Linear Units (ELU)

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$



(i) Solved the dead neuron issue of ReLU

(ii) Advantages
- a) all advantages of ReLU
- b) No Dead ReLU issue
- c) mean of output is close to 0 ⇒ zero-centered

(iii) Disadvantages
- a) Slightly more computationally intensive
- b) Although it is theoretically better ~~than~~ than ReLU, but no good practical evidence.

☆ GLU (Gated Linear Units)

Paper: Language Modeling with Gated Convolutional Networks

Working

(1) To activate a layer of dimension d, we make it output double its dimension ($2 \times d$)

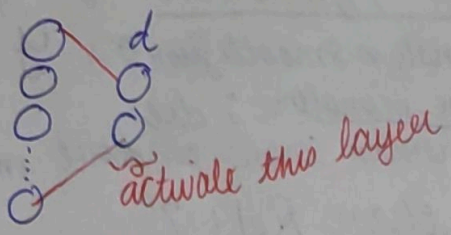
(2) Split it into 2 halves. First half = original layer output
Second half = gating layer.

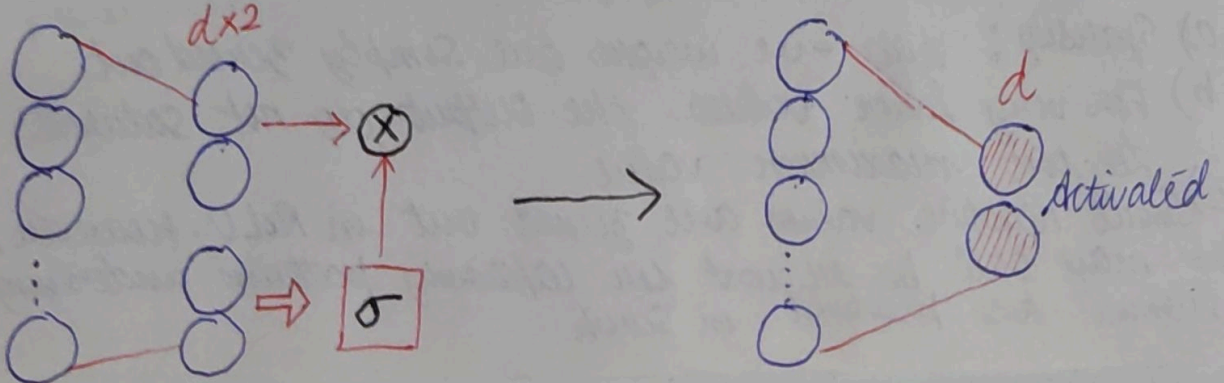
(3) Gated layer is followed by a sigmoid activation function which squashes each value to a range of 0-1.

(4) Gate values are then element-wise multiplied with first half.

$$(xW_1 + b_1) \otimes \text{Sigmoid}\,(xW_2 + b_2)$$

Example $d = 2$





★ Softplus

a) Similar to the ReLU
(but relatively smoother)

b) At 0 it is smooth & differentiable

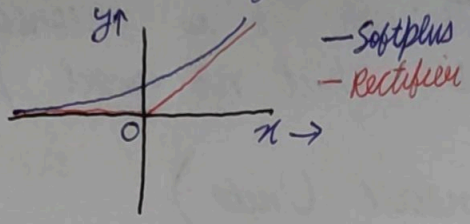
c) derivative of softplus func^n = logistic func^n

$$f(x) = \ln(1+e^x)$$

$$\Rightarrow \frac{df(x)}{dx} = \frac{e^x}{1+e^x}$$

$$= \frac{1}{1+e^{-x}} \quad \text{(logistic func}^n\text{)}$$

d) Wide acceptance range $(0, +\infty)$



★ Maxout

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

a) generalizes ReLU & Leaky ReLU
(both ReLU & Leaky ReLU are special cases)

b) All benefits of ReLU eg no saturation
& none of the drawbacks of ReLU

★ Swish (A Self-Gated) function

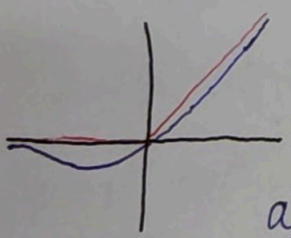
$y = x \cdot \text{sigmoid}(x)$

$y = x \,\text{sigmoid}(x)$

$= x * (1+e^{-x})^{-1}$

(i) Swish is smooth func^n

(ii) non-monotonic: does not remain stable as move in 1 direction.



Why better than ReLU?

a) Sparsity: Very -ve weights are simply zeroed out

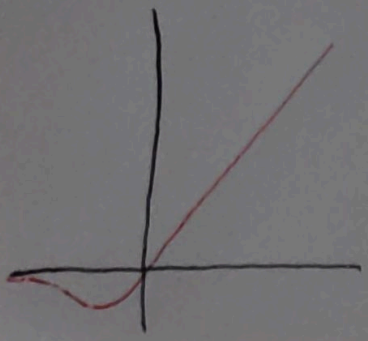
b) For very large values, the outputs do not saturate to the maximum value

c) Small negative values are zeroed out in ReLU. However, those -ve values may still be relevant for capturing patterns underlying in the data, which are preserved in Swish.

★ Mish Activation

$f(x) = x \tanh(\ln(1+e^x))$

code: `x * (torch.tanh(F.softplus(x)))`



(i) A modified gated form of Softplus Activation function

(ii) Advantages

a) Unbounded above → no saturation

b) Small -ve values are not zero
⇒ better gradient flow

c) Continuous
↳ Smooth at 0, unlike ReLU.
⇒ effective optimization & generalization

Results of Mish

Model	Mish	Swish	ReLU
ResNet v2-110	74.41%	74.13%	73%
WRN 22-10	72.32%	71.89%	72.2%
WRN 40-4	69.52%	69.59%	69.35%
DenseNet -121	66.31%	65.91%	65.50%
DenseNet -169	65.38%	65.69%	64.99%
ResNext -50	67.58%	66.72%	67.52%
MobileNet v1	50.09%	49.95%	49.20%
SE Net-18	64.38%	63.89%	62.71%
Shuffle Net V2	59.35%	58.91%	58.56%
Squeeze Net	63.07%	62.11%	60.92%