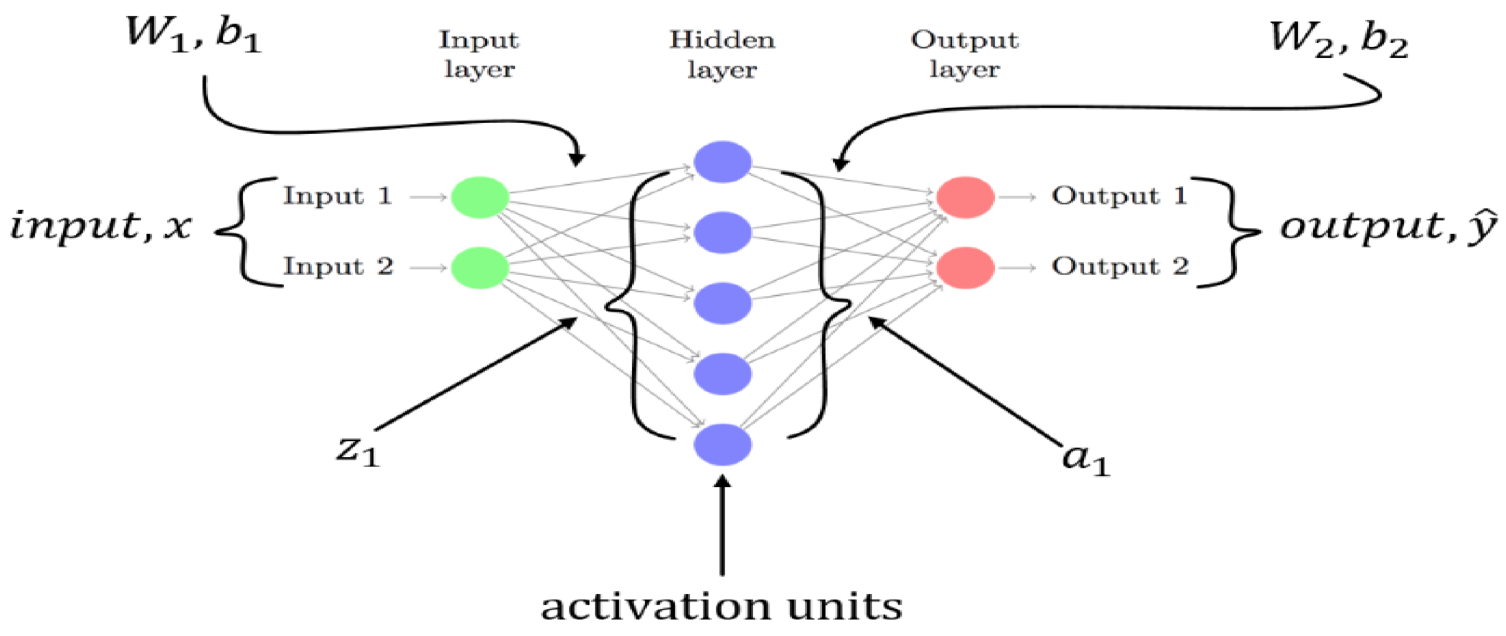
DNN_01

Activation Function in NN

3 Layer NN Architecture



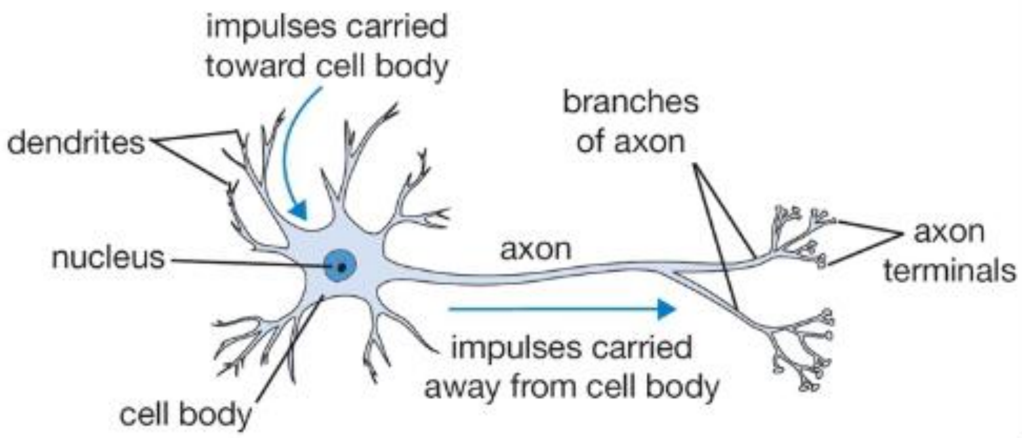
$$z_1 = xW_1 + b_1$$
$$a_1 = z_1$$
$$z_2 = a_1 W_2 + b_2$$
$$a_2 = \hat{y} = softmax(z_2)$$

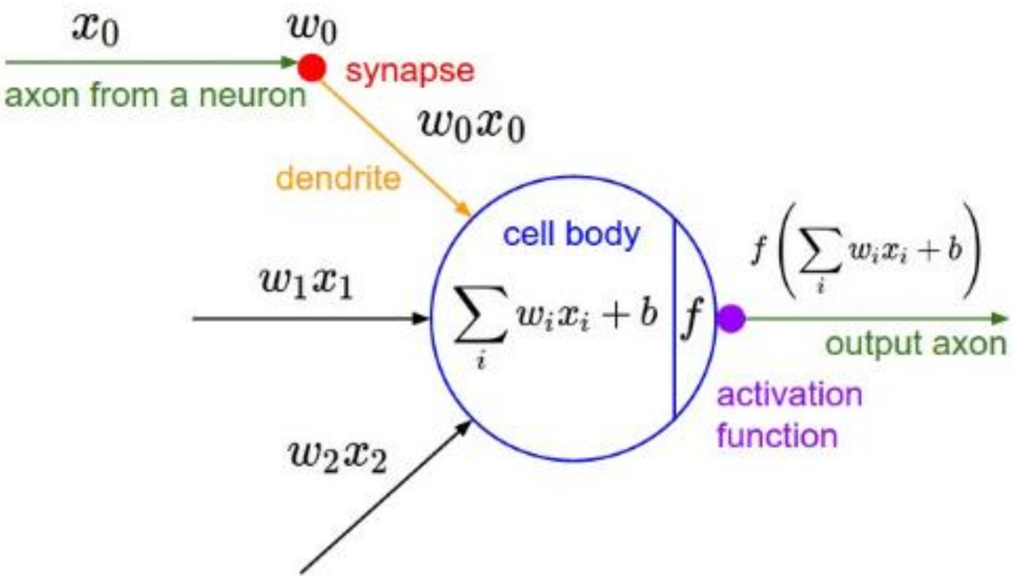
$$\delta_3 = \hat{y} - y, \delta_2 = \delta_3 W_2^T$$
$$\frac{\partial L}{\partial W_2} = a_1^T \delta_3 \qquad \frac{\partial L}{\partial b_2} = \delta_3$$
$$\frac{\partial L}{\partial W_1} = x^T \delta_2 \qquad \frac{\partial L}{\partial b_1} = \delta_2$$

Neural Network Training Loop :SGD

- Loop :
  - X,Y = Sample batch of data
  - loss = forward(X,Y)
  - gradient = backward(loss)
  - update_param(gradient)

Neuron Model



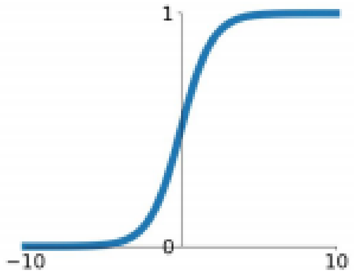


Each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function)

Activation functions

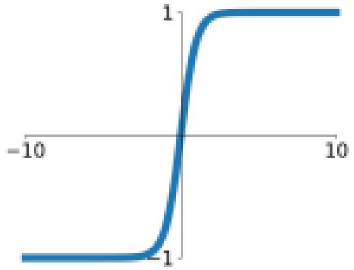
Sigmoid

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



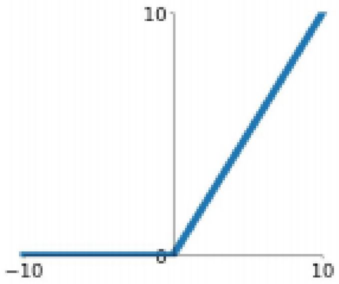
tanh

$\tanh(x)$



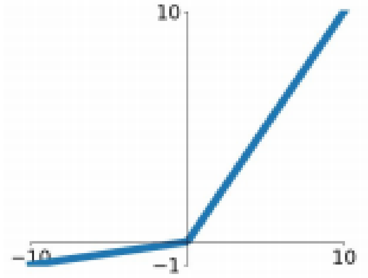
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

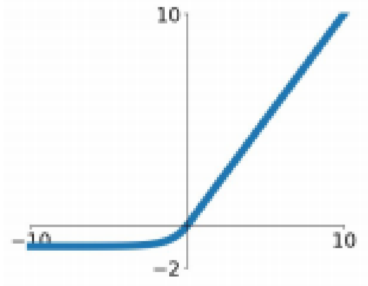


Maxout

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

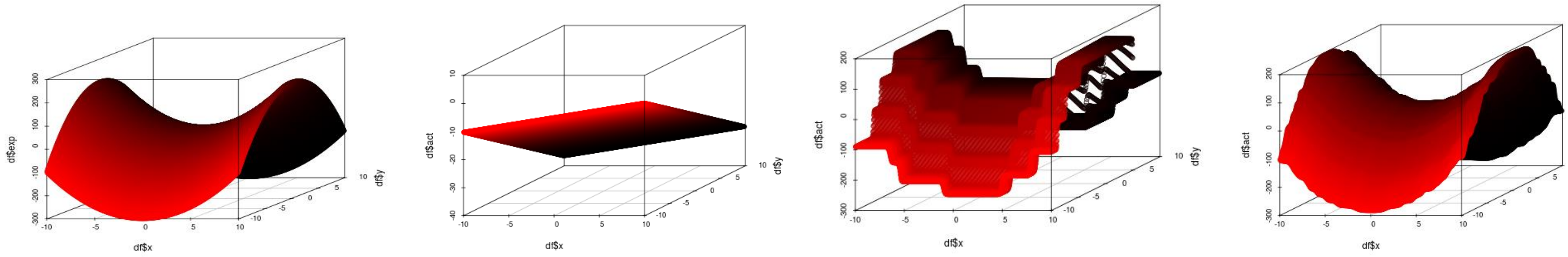
ELU

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

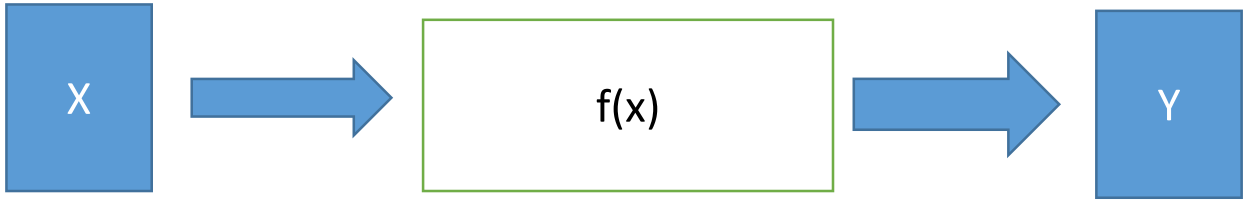


Why to use Activation Functions.

- To make NN a Universal Function Approximator.
  - By adding Non-Linearity
  - We need a Neural Network Model to learn and represent almost anything and any arbitrary complex function which maps inputs to outputs.
  - It means that they can compute and represent any function.



a) b) c) d)

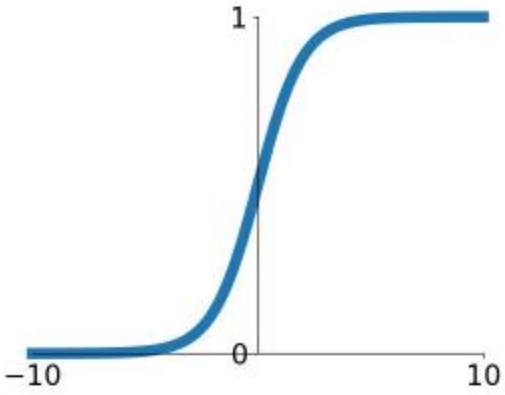

X
f(x)
Y

Properties of Activation

- Non-Linearity
- Range
- Continuously Differentiable
- Monotonic
- Computational Cost

Activation Function : Sigmoid

- Derivative : z(1-z) , z=sigmoid(x)
- Squashes input to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

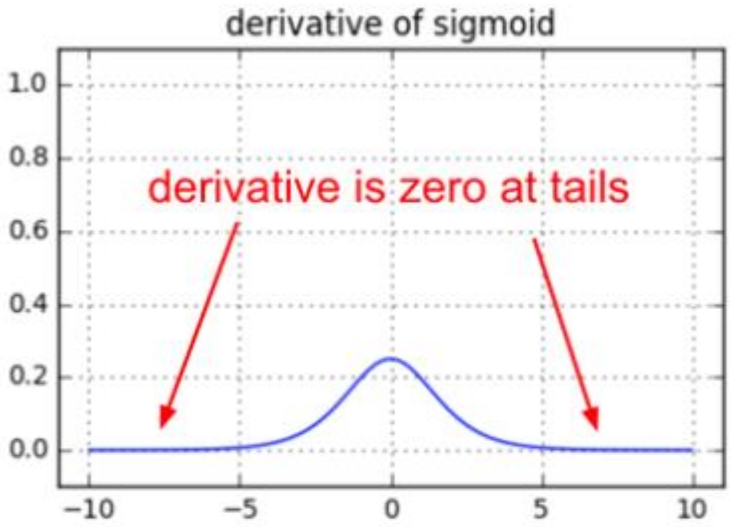


Sigmoid

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

Problems

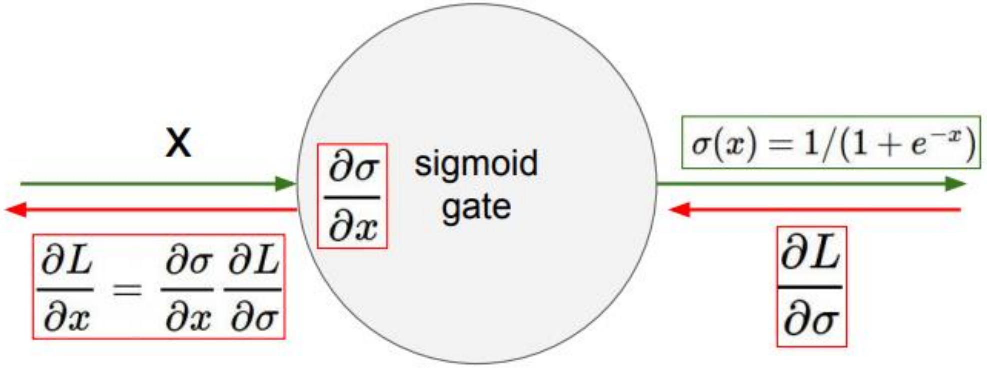
- exp() is a bit compute expensive.
- Saturated neurons kills gradient
- Sigmoid output is not zero centered
- Slow convergence

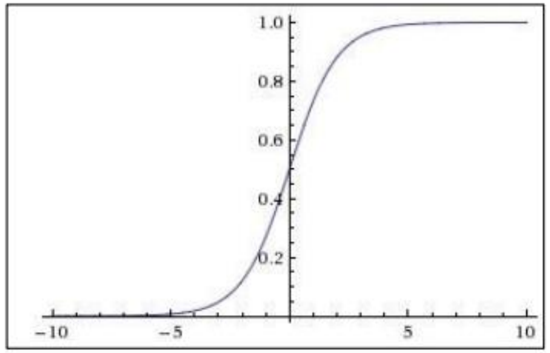


Activation Function : Sigmoid

- Saturated neurons kills gradient
- Derivative : z(1-z)
- If z=1 or 0, dz = 0

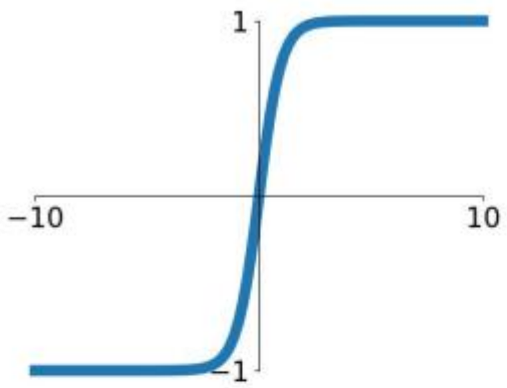
- Initial layer learn much slowly compared to end layers.





What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Activation Function : tanh(x)



tanh(x)

- Derivative : z(1-z^2) , z=tanh(x)
- Squashes input to range [-1,1]
- zero centered

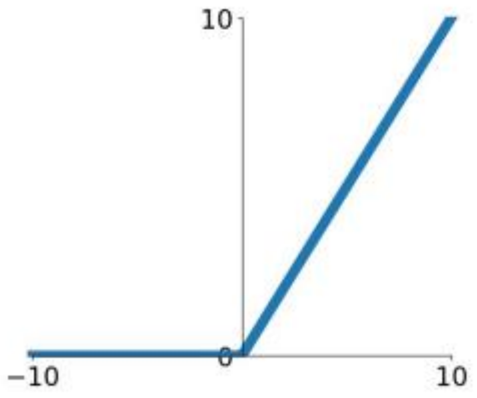
Problems

- exp() is a bit compute expensive.
- saturated neurons kills gradient.

Activation Function : ReLU

f(x) = max(0,x)

- Derivative : 1,if x>= 0 else 0
- Range : [0,inf]
- Does not saturate (in + region)
- Computationally very efficient
- Converges much faster than sigmoid/tanh (6x)
- Gradient through ReLU unit remain large when active

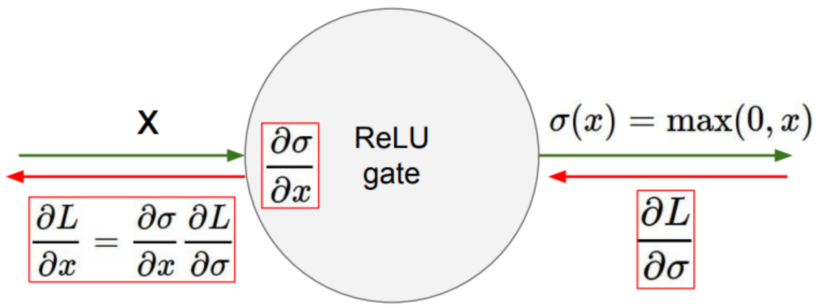


ReLU
(Rectified Linear Unit)

Problems

- Not zero centered
- Neuron dies sometimes
- Be careful with your learning rates

Activation Function : ReLU



- Neuron dies sometimes:

  ReLU neurons output zero and have zero derivatives for all negative inputs.
  So, if the weights in your network always lead to negative inputs into a ReLU neuron, that neuron will not activate , hence no updates , it dies forever.

  A large learning rate amplifies this problem.

if x < 10 : z = 0 , dz = 0
if x = 0 : z= 0 , dz = 1
if x> 10 : z = 10 , dz = 1

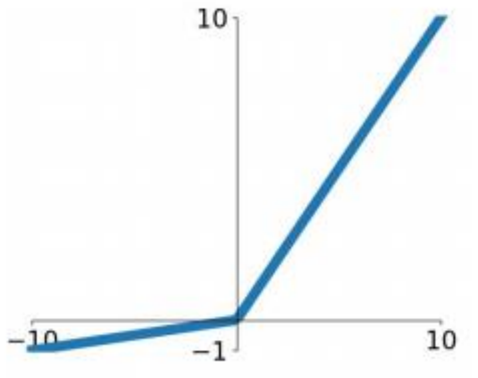
Activation Function : Leaky ReLU

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

- Derivative : 1,if x> 0 else 0.01
- Does not saturate (in + region)
- Computationally very efficient
- Converges much faster than sigmoid/tanh (6x)
- Will not "die".

Parametric Rectifier (PReLU):

F(x) = max(ax , x)



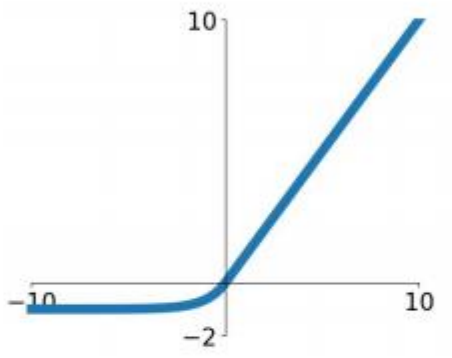
Leaky ReLU

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

Activation Function :
Exponential Linear Units (ELU)

- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

- Computation requires exp()



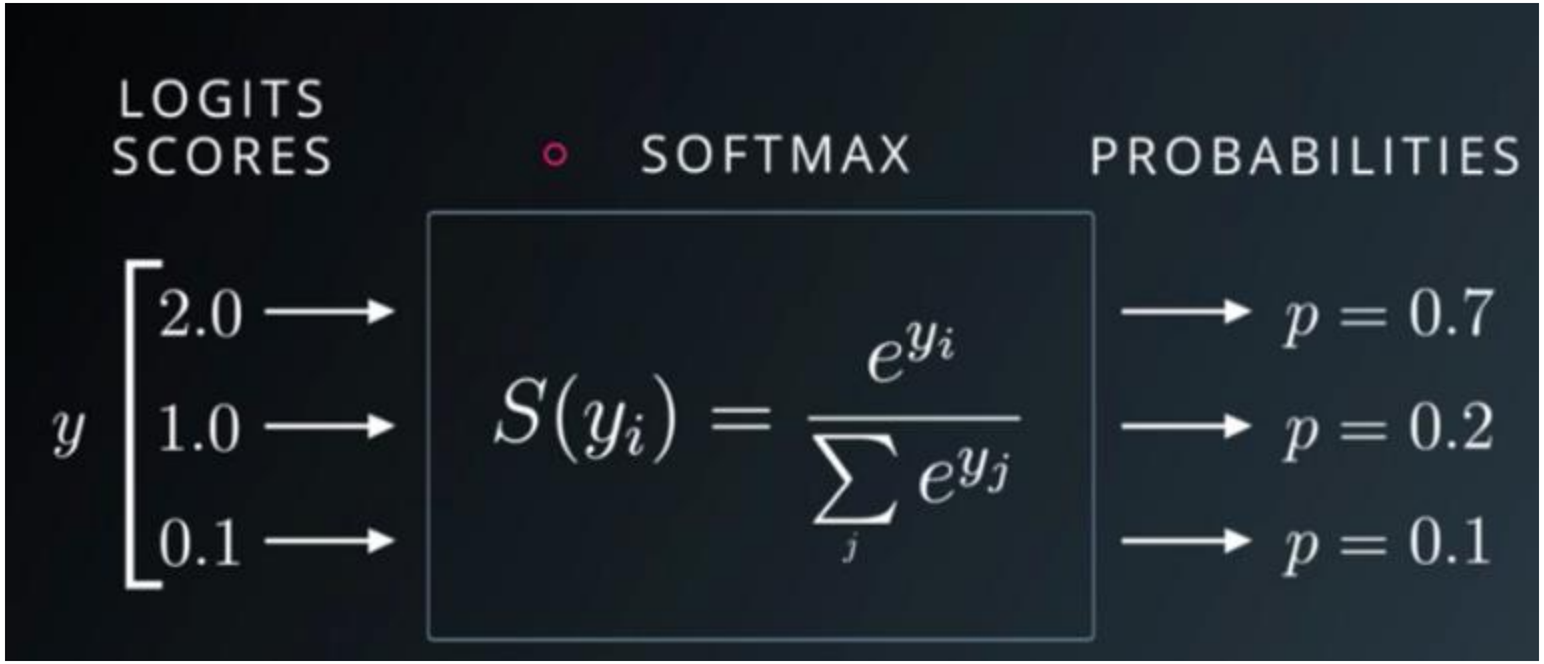
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

Quick summary

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

Softmax

- It converts logits into probabilities values
- Used in multi-class classification problem
- Sum of all output probabilities is 1
- Push one result closer to 1 while another closer to 0.



Reference:

- www.towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f
- www.deeplearningbook.org