AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

19CSE437
DEEP LEARNING FOR COMPUTER VISION
L-T-P-C: 2-0-3-3

Feed Forward Neural Networks

- Optimization – Hyper Parameter Tunings
  - Activation Functions
  - Xavier-He Initialisation

Citation Note: content, of this presentation were inspired by the awesome lectures and the material offered by Prof. ***Mitesh M. Khapra*** *on NPTEL's Deep Learning course*

Feed Forward NN - Hyper Parameter Tunings

Network Architectures

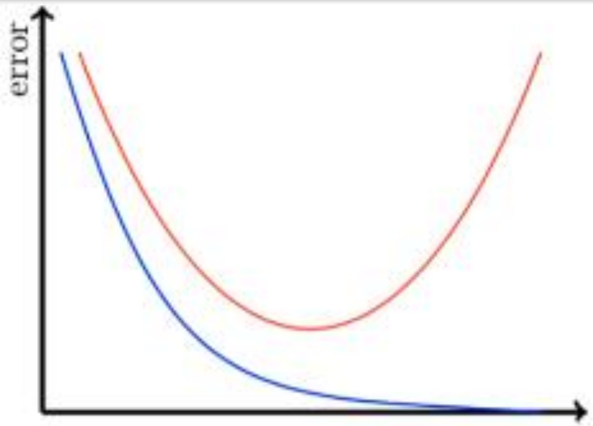
- Number of layers
- Number of neurons

Algorithms

- Vanilla/Momentum /Nesterov GD
- AdaGrad
- RMSProp
- Adam

Activation Functions

- tanh (RNNs)
- relu (CNNs, DNNs)
- leaky relu (CNNs)



Strategies

- Batch
- Mini-Batch (32, 64, 128)
- Stochastic
- Learning rate schedule

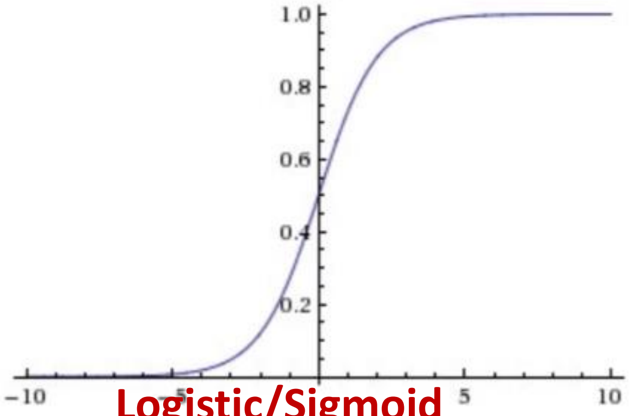
Regularization

- L2
- Early stopping
- Dataset augmentation
- Drop-out
- Batch Normalizat

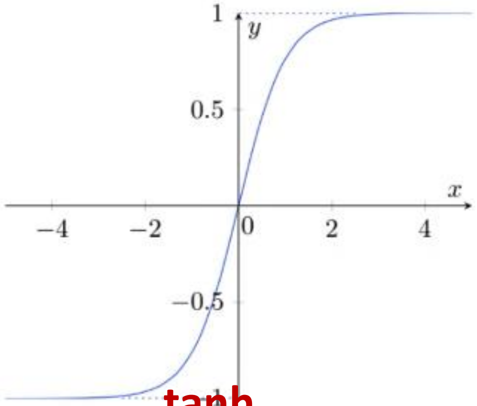
Initialization Methods

- *Xavier*
- *He*

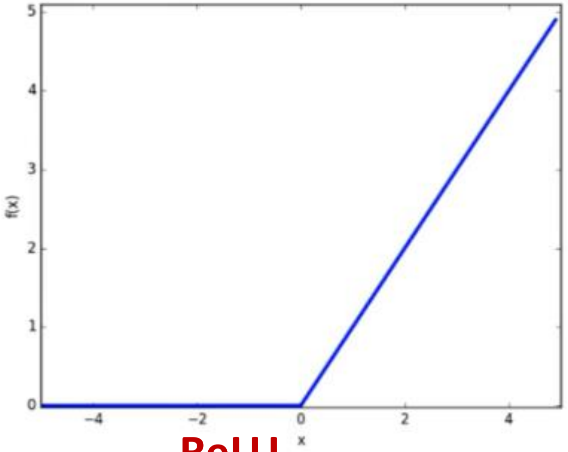
Activation functions



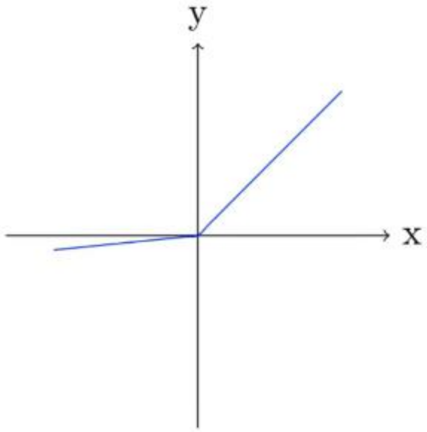
Logistic/Sigmoid



tanh



ReLU



Leaky ReLU

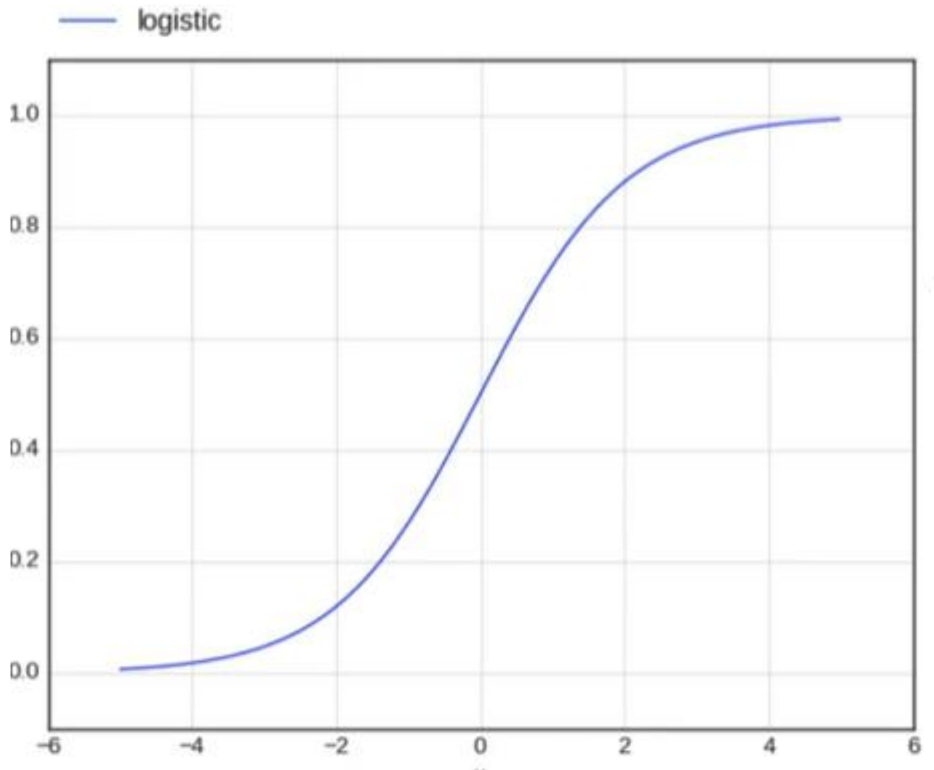
Logistic/Sigmoid

Vanishing Gradient problem

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

$$f'(x) = \frac{\partial f(x)}{\partial x} = f(x) * (1 - f(x))$$

- always normalize the inputs (so that they lie between 0 to 1)



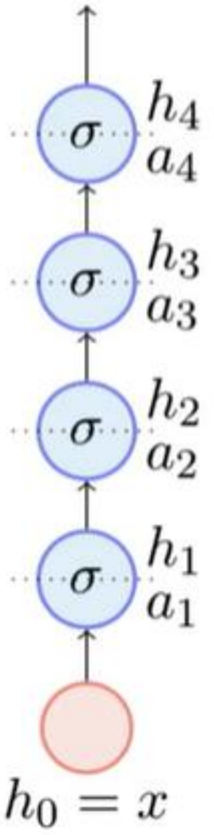
- Saturated neurons cause the gradients to vanish

Saturation:

$$\text{when } f(x) = 0 \text{ or } 1$$

$$\text{and hence } f'(x) = 0$$

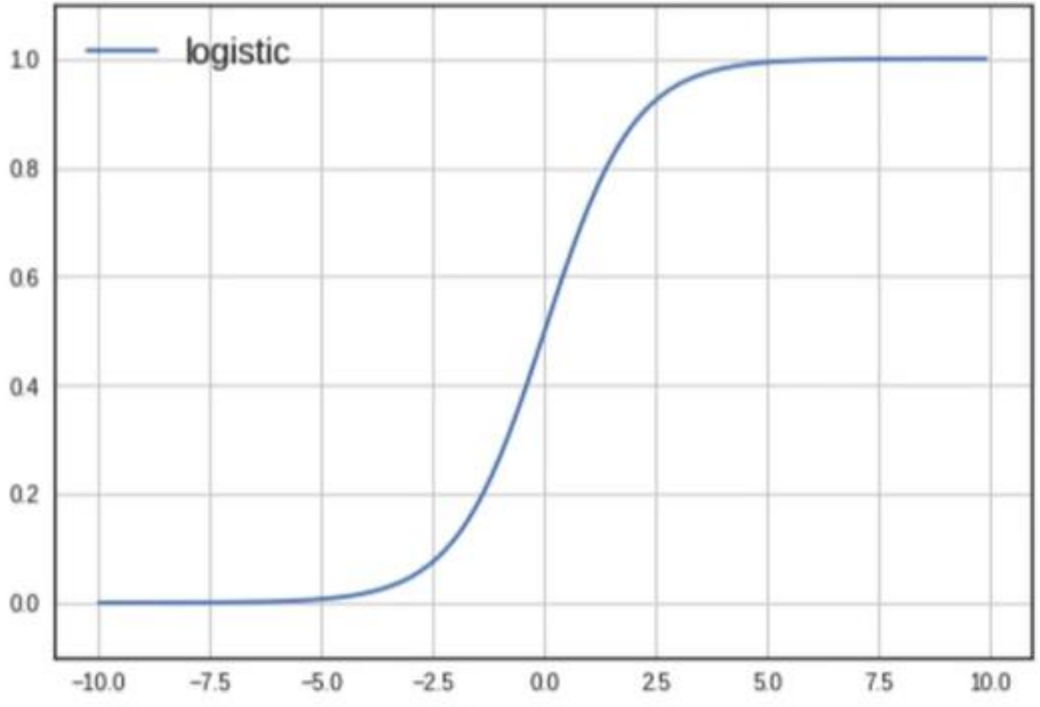
$$\underbrace{\frac{\partial \mathscr{L}(\theta)}{\partial W_{111}}}_{\text{Talk to the weight directly}} = \underbrace{\frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3}}_{\text{Talk to the output layer}} \underbrace{\frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_1}{\partial W_{111}}}_{\text{and now talk to the weights}}$$



$\frac{\partial h_2}{\partial a_2}$=0 Then ∇w = 0 and hence no updation in weight happens

Gradient at some neurons vanishes and no more learning happens- Vanishing gradient

Logistic/Sigmoid

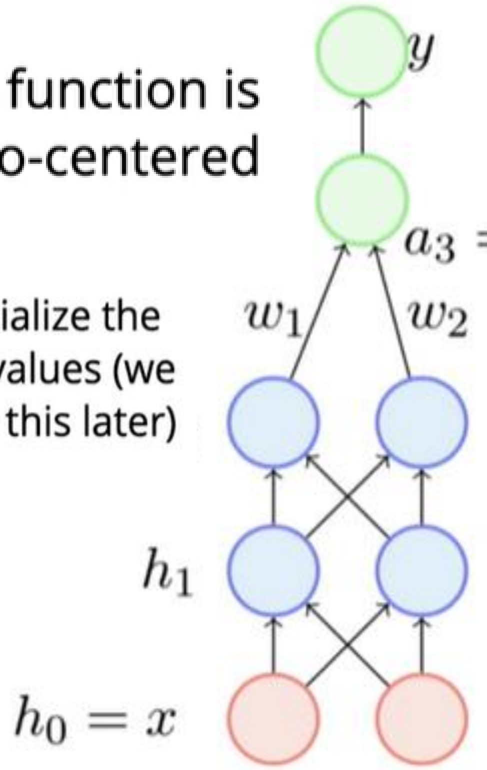


$\sum_{i=1}^{4} w_i x_i$

Logistic Neuron is not zero centered which restricts the movement of gradient on training – takes time to converge

logistic function is not zero-centered

Remember to initialize the weights to small values (we will come back to this later)



$a_3 = w_1 * h_{21} + w_2 * h_{22}$

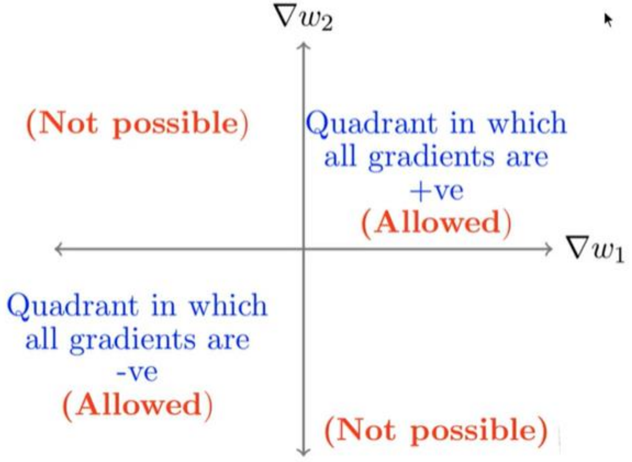
$$\nabla w_1 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} \frac{\partial a_3}{\partial w_1}$$

$$\nabla w_2 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} \frac{\partial a_3}{\partial w_2}$$

+/- +

$$\nabla w_1 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} h_{21}$$

$$\nabla w_2 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} h_{22}$$



∇w_1 :

∇w_2 :

Will be either all + or all-

+/- +

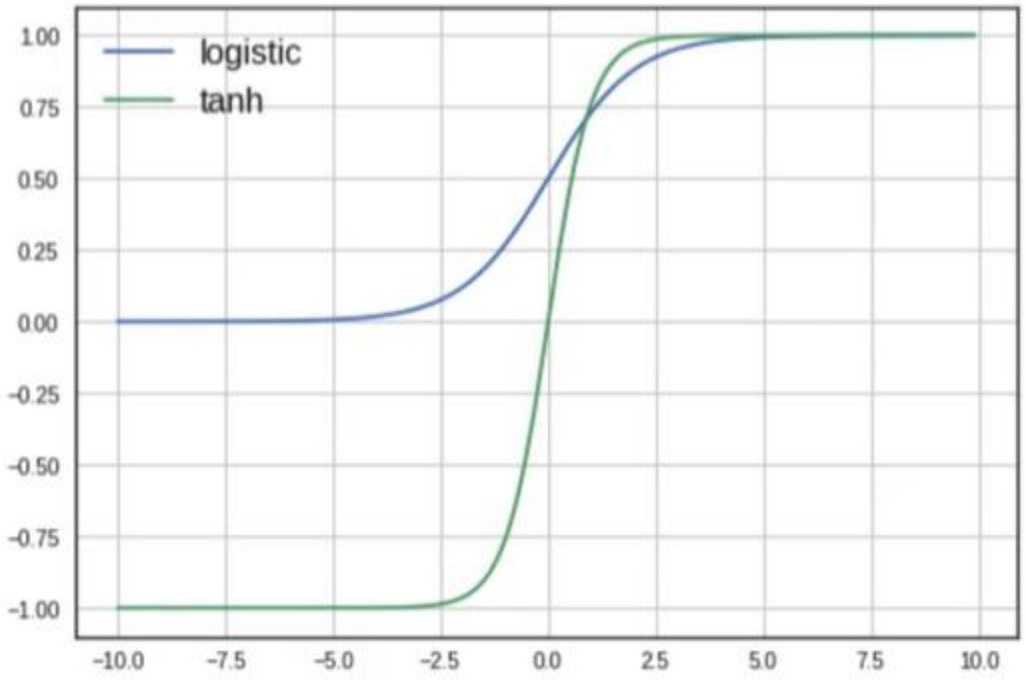
The gradients w.r.t. all the weights connected to the same neuron are either all +ve or all -ve

Better Activation Functions- tanh

- ✗ Saturated logistic neurons cause the gradients to vanish
- ✗ logistic function is not zero-centered
- ✗ logistic function is computationally expensive (because of e^x

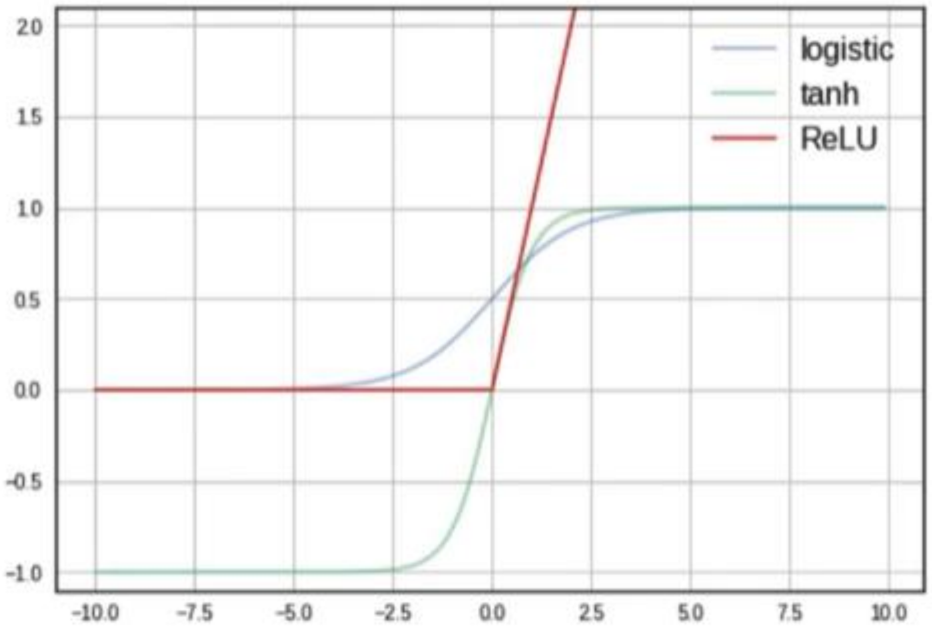
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f'(x) = \frac{\partial f(x)}{\partial x} = (1 - (f(x))^2)$$



- ✗ Saturated tanh neurons cause the gradients to vanish
- ✓ tanh is zero-centered
- ✗ tanh is computationally expensive (because of e^x

Better Activation Functions- ReLU (Rectified Linear activation Unit)

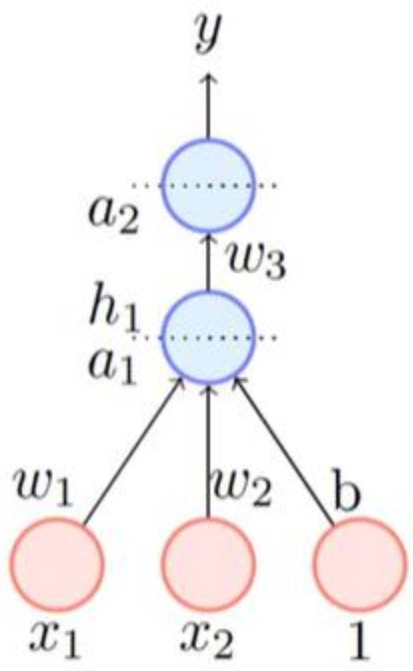


$$f(x) = max(0, x)$$

$$f'(x) = \frac{\partial f(x)}{\partial x} = \begin{cases} 0 & if \quad x < 0 \\ 1 & if \quad x > 0 \end{cases}$$

- ✔ Does not saturate in the positive region
- ✘ Not zero-centered
- ✔ Easy to compute (no expensive e^x)

Issues with ReLU



$$h_1 = ReLU(a_1) = max(0, a_1)$$
$$= max(0, w_1 x_1 + w_2 x_2 + b)$$

What happens if b takes on a large negative value due to a large negative update (∇b) at some point ?

$$w_1 x_1 + w_2 x_2 + b < 0 \quad [if \quad b << 0]$$
$$\implies h_1 = 0 \quad [dead\ neuron]$$
$$\implies \frac{\partial h_1}{\partial a_1} = 0$$

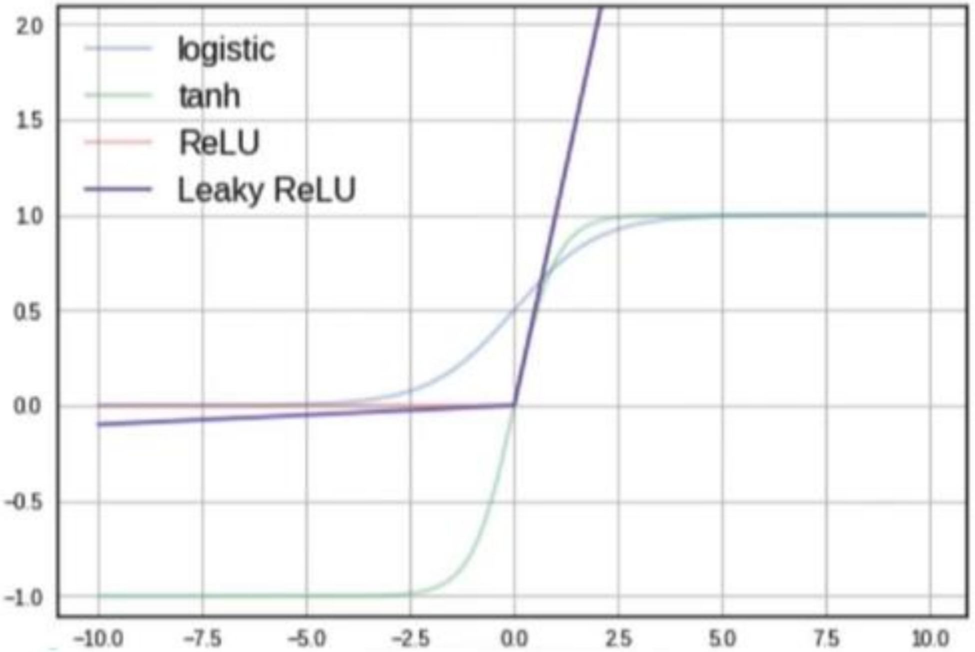
$$\nabla w_1 = \frac{\partial \mathscr{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

- A large fraction of ReLU units can die if the learning rate is set too high

- It is advised to initialize the bias to a positive value

- Use other variants of ReLU

Leaky ReLU

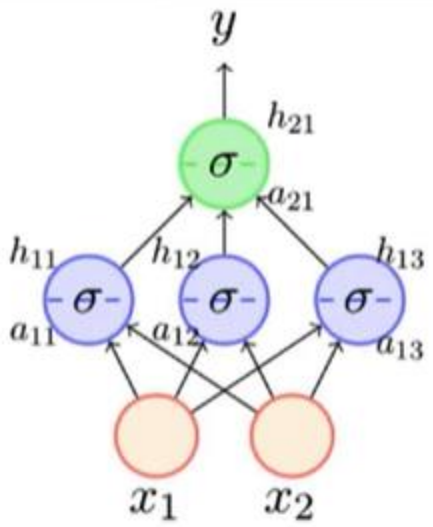


- Does not saturate in positive or negative region
- Will not die (0.01 x ensures that at least a small gradient will flow through)

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

$$f'(x) = \frac{\partial f(x)}{\partial x} = \begin{cases} 0.01 & if \quad x < 0 \\ 1 & if \quad x > 0 \end{cases}$$

Initialization of Weights and Bias



$$a_{11} = w_{11}x_1 + w_{12}x_2$$
$$a_{12} = w_{21}x_1 + w_{22}x_2$$
$$\therefore a_{11} = a_{12} = 0$$
$$\therefore h_{11} = h_{12}$$

Initialise w, b
Iterate over data:
$\quad$ *compute* $\hat{y}$
$\quad$ *compute* $\mathscr{L}(w, b)$
$\quad$ $w_{11} = w_{11} - \eta \Delta w_{11}$
$\quad$ $w_{12} = w_{12} - \eta \Delta w_{12}$
$\quad$ ... ...
till satisfied

$$\nabla w_{11} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} . x_1$$
$$\nabla w_{21} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} . x_1$$
$$but \quad h_{11} = h_{12}$$
$$and \quad a_{12} = a_{12}$$
$$\therefore \nabla w_{11} = \nabla w_{21}$$

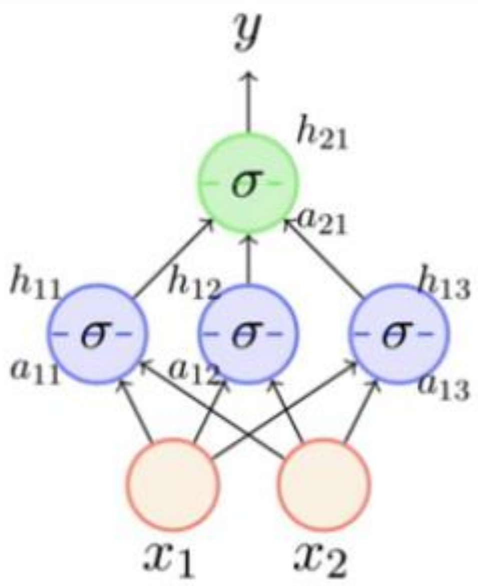
✔ never initialize all weights to same value

✘ This symmetry will never break during training (**symmetry breaking problem**)

✘ Hence **weights** connected to the same neuron should **never be initialized to the same value**

✔ never initialize all weights to 0

Initialization of Weights and Bias

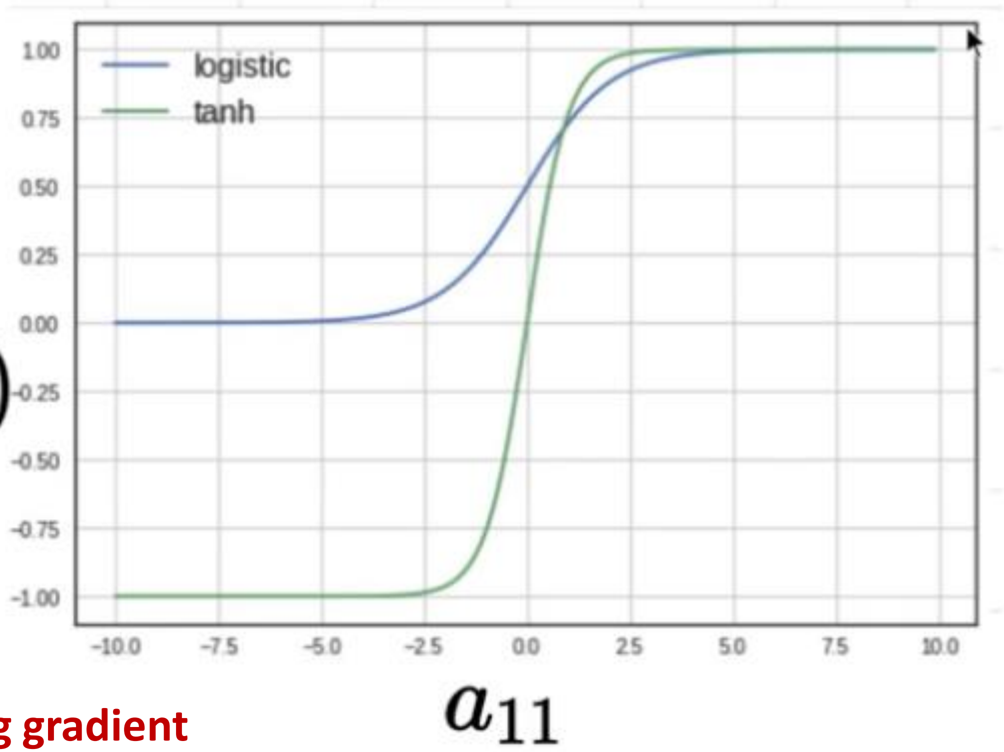


$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

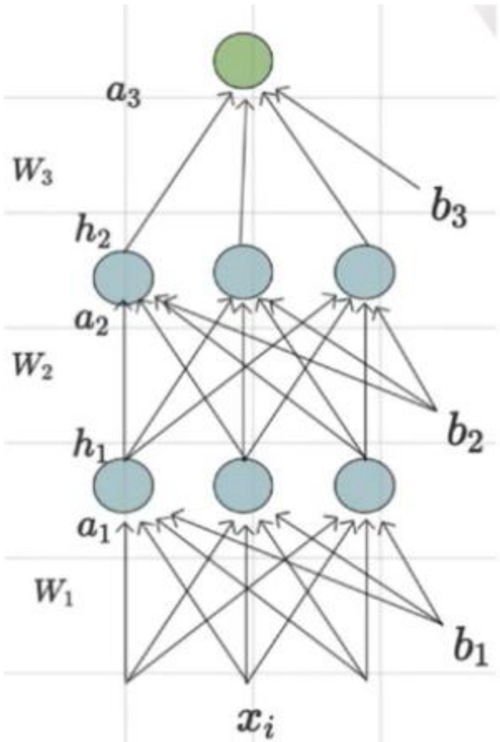
- always normalize the inputs (so that they lie between 0 to 1)
- never initialize weights to large values

$$h_{11} = \sigma(a_{11})$$



Blowing up a11 cause vanishing gradient

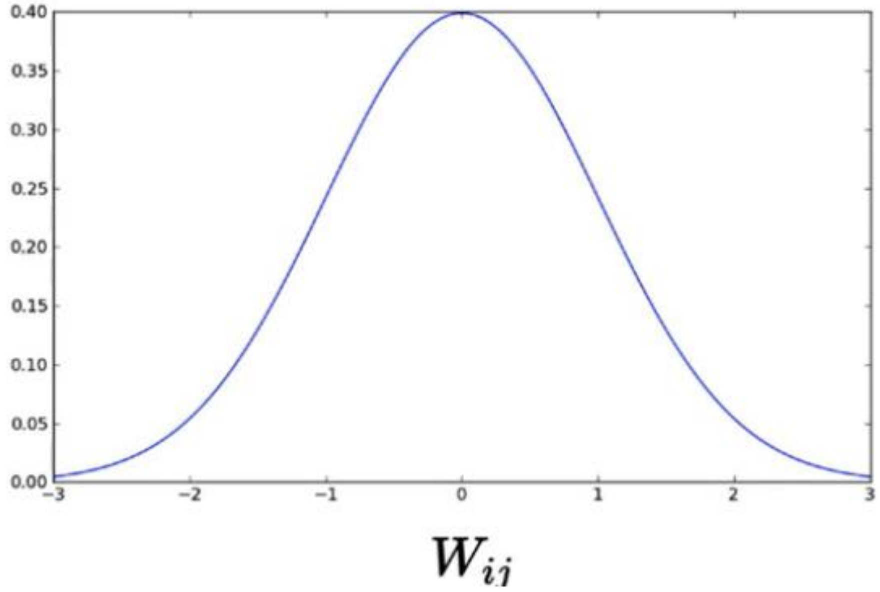
Xavier Initialization – Softmax and tanh activation



$$a_2 = W_2 x + b_2$$

$$h_2 = \sigma(a_2)$$

Initializing weights from
a *uniform* distribution in [-1,1]
and then scaling by 1/√*n*



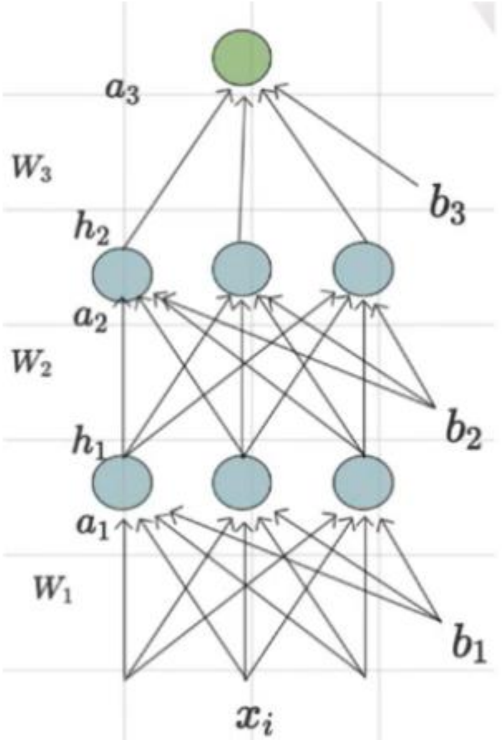
Xavier initialization sets a layer's weights to values chosen from a random uniform distribution that's bounded between

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$$

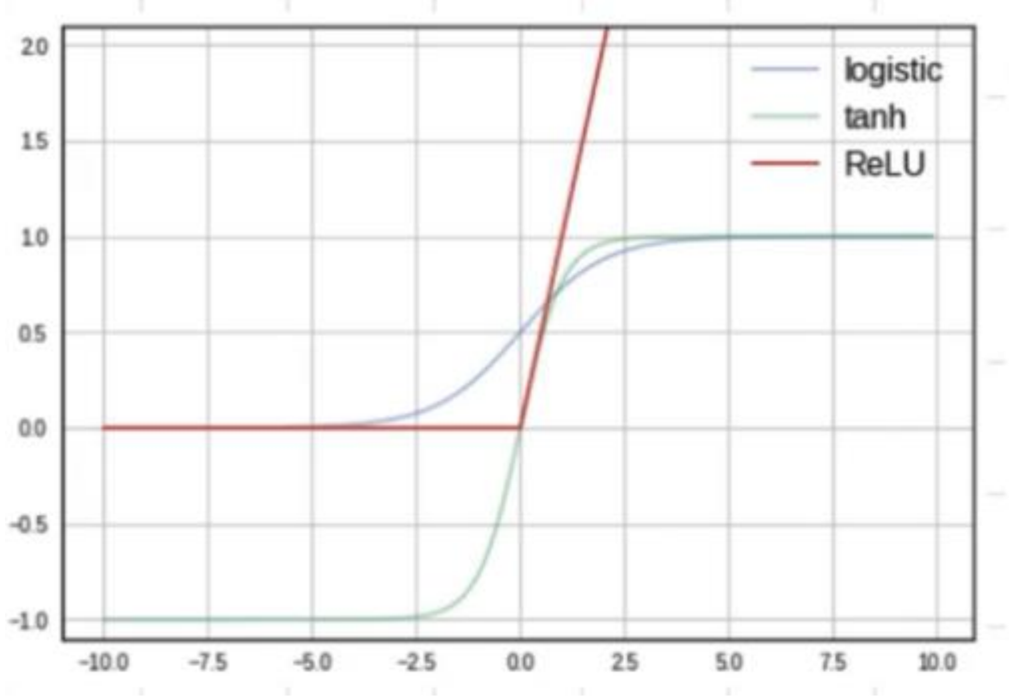
where n_i is the number of incoming network connections, or "fan-in," to the layer, and n_{i+1} is the number of outgoing network connections from that layer, also known as the "fan-out."

He Initialization ReLu activation

The **he** initialization method is calculated as a random number with a Gaussian probability distribution (G) with a mean of 0.0 and a standard deviation of $\sqrt{\frac{2}{n}}$, where *n* is the number of inputs to the node.



$$a_2 = W_2 x + b_2$$
$$h_2 = \sigma(a_2)$$



In ReLU half the neurons are dead hence n/2

Namah Shivaya