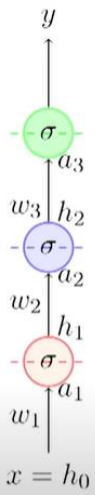
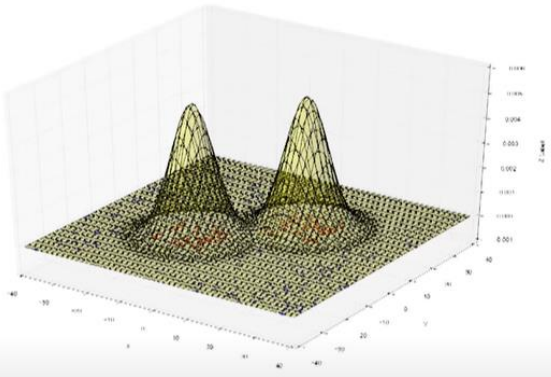
- Before we look at activation functions, let's try to answer the following question: "What makes Deep Neural Networks powerful ?"



- Consider this deep neural network
- Imagine if we replace the sigmoid in each layer by a simple linear transformation

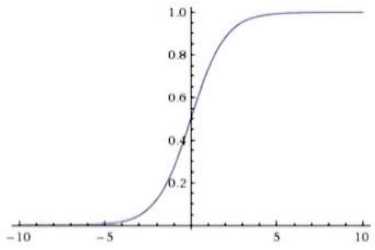
$$y = (w_4 * (w_3 * (w_2 * (w_1 x))))$$

- Then we will just learn y as a linear transformation of x
- In other words we will be constrained to learning linear decision boundaries
- We cannot learn arbitrary decision boundaries



- In particular, a deep linear neural network cannot learn such boundaries
- But a deep non linear neural network can indeed learn such boundaries (recall Universal Approximation Theorem)

- Now let's look at some non-linear activation functions that are typically used in deep neural networks (Much of this material is taken from Andrej Karpathy's lecture notes [1])



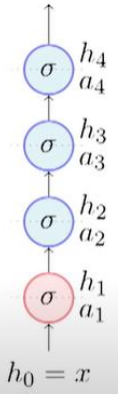
Sigmoid

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- As is obvious, the sigmoid function compresses all its inputs to the range [0,1]
- Since we are always interested in gradients, let us find the gradient of this function

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

(you can easily derive it)

- Let us see what happens if we use sigmoid in a deep netv

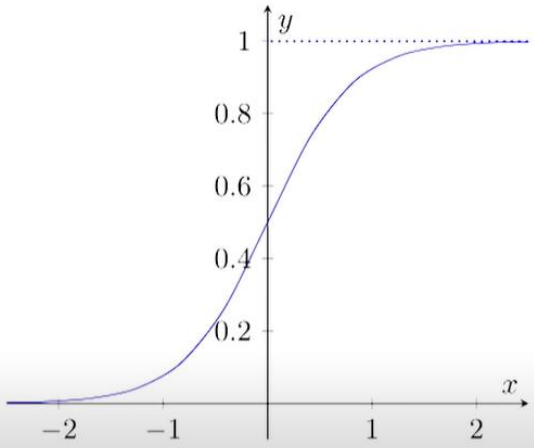


$$a_3 = w_2 h_2$$
$$h_3 = \sigma(a_3)$$

- While calculating ∇w_2 at some point in the chain rule we will encounter

$$\frac{\partial h_3}{\partial a_3} = \frac{\partial \sigma(a_3)}{\partial a_3} = \sigma(a_3)(1 - \sigma(a_3))$$

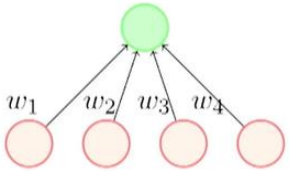
- What is the consequence of this ?
- To answer this question let us first understand the concept of saturated neuron ?



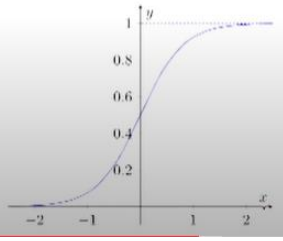
- A sigmoid neuron is said to have saturated when $\sigma(x) = 1$ or $\sigma(x) = 0$
- What would the gradient be at saturation?
- Well it would be 0 (you can see it from the plot or from the formula that we derived)

Saturated neurons thus cause the gradient to vanish.

Saturated neurons thus cause the gradient to vanish.



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



- But why would the neurons saturate ?
- Consider what would happen if we use sigmoid neurons and initialize the weights to very high values ?
- The neurons will saturate very quickly
- The gradients will vanish and the training will stall (more on this later)

- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered
- Consider the gradient w.r.t. w_1 and w_2

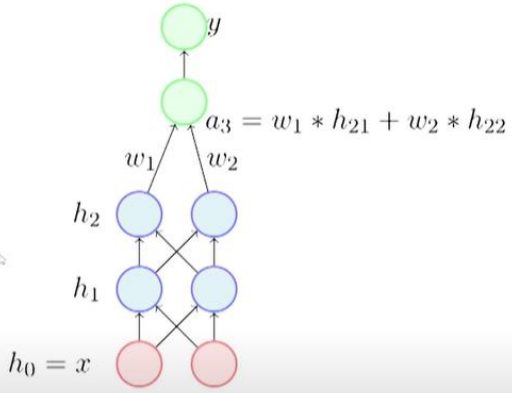
$$\nabla w_1 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} h_{21}$$

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

- Note that h_{21} and h_{22} are between $[0, 1]$ (*i.e.*, they are both positive)
- So if the first common term (in red) is positive (negative) then both ∇w_1 and ∇w_2 will be positive (negative)

12:07 / 28:08

- Why is this a problem??

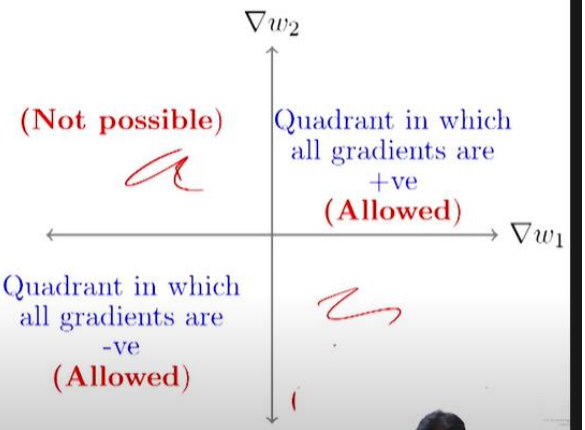


- Essentially, either all the gradients at a layer are positive or all the gradients at a layer are negative

- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered

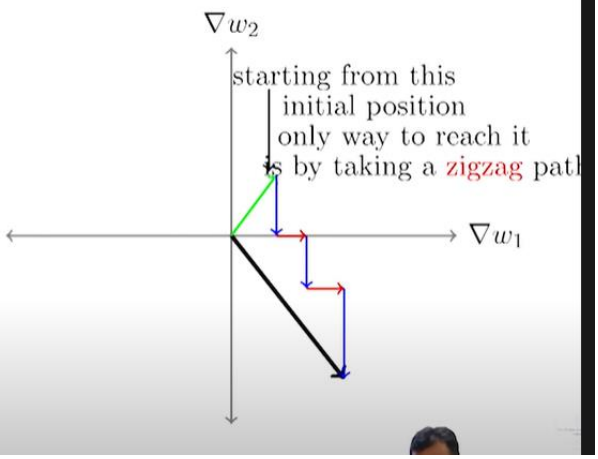
$\theta = \theta - \eta \nabla \theta$

- This restricts the possible update directions

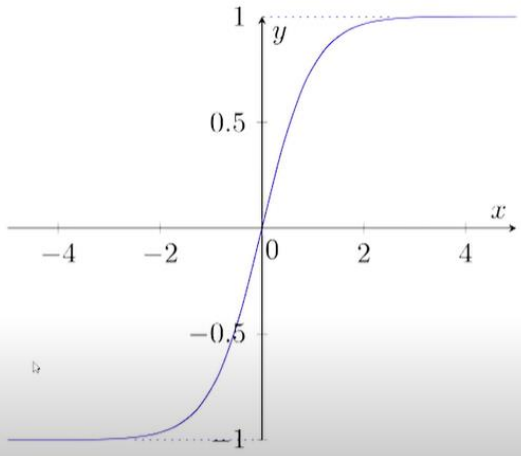


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered

- And lastly, sigmoids are computationally expensive (because of $\exp(x)$)



tanh(x)



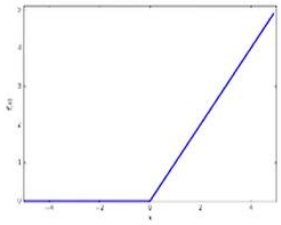
f(x) = tanh(x)

- Compresses all its inputs to the range [-1,1]
- Zero centered
- What is the derivative of this function?

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

- The gradient still vanishes at saturation
- Also computationally expensive

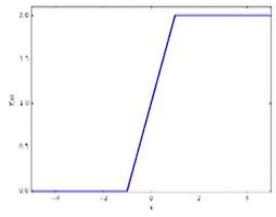
ReLU



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

- Is this a non-linear function?

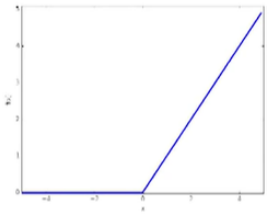
ReLU



$$f(x) = max(0, x + 1) - max(0, x - 1)$$

- Is this a non-linear function?
- Indeed it is!
- In fact we can combine two ReLU units to recover a piecewise linear approximation of the sigmoid function

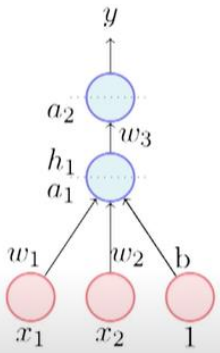
ReLU



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

Advantages of ReLU

- Does not saturate in the positive region
- Computationally efficient
- In practice converges much faster than $sigmoid/tanh$[1]

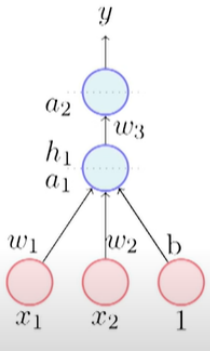


- In practice there is a caveat
- Let's see what is the derivative of ReLU(x)

$$\frac{\partial ReLU(x)}{\partial x} = 0 \quad if \quad x < 0$$
$$= 1 \quad if \quad x > 0$$

- Now consider the given network
- What would happen if at some point a large gradient causes the bias b to be updated to a large negative value?

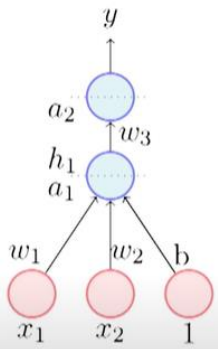
$$w_1 x_1 + w_2 x_2 + b < 0 \quad [if \quad b << 0]$$



- The neuron would output 0 [dead neuron]
- Not only would the output be 0 but during backpropagation even the gradient $\frac{\partial h_1}{\partial a_1}$ would be zero
- The weights w_1, w_2 and b will not get updated [$\because$ there will be a zero term in the chain rule]

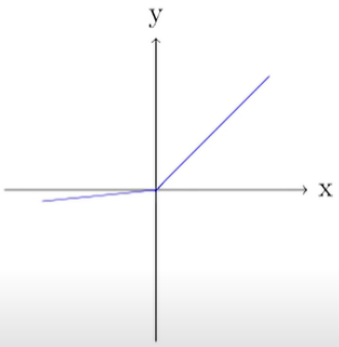
$$\nabla w_1 = \frac{\partial \mathscr{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

- The neuron will now stay dead forever!!



- In practice 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 (0.01)
- Use other variants of ReLU (as we will soon see)

Leaky ReLU



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

- No saturation
- Will not die ($0.01x$ ensures that at least a small gradient will flow through)
- Computationally efficient
- Close to zero centered ouputs

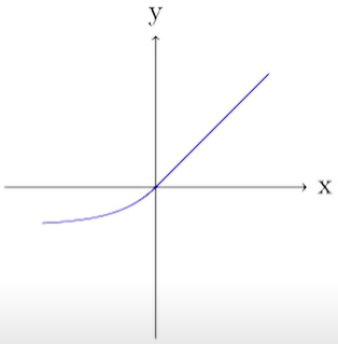
Parametric ReLU

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

α *is a parameter of the model*

α *will get updated durir*

Exponential Linear Unit



$$f(x) = x \quad if \quad x > 0$$
$$= ae^x - 1 \quad if \quad x \leq 0$$

- All benefits of ReLU
- $ae^x - 1$ ensures that at least a small gradient will flow through
- Close to zero centered outputs
- Expensive (requires computation of exp(x))

Maxout Neuron

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

- Generalizes ReLU and Leaky ReLU
- No saturation! No death!
- Doubles the number of parameters

Things to Remember

- Sigmoids are bad
- ReLU is more or less the standard unit for Convolutional Neural Networks
- Can explore Leaky ReLU/Maxout/ELU
- tanh sigmoids are still used in LSTMs/RNNs (we will see more on this later)