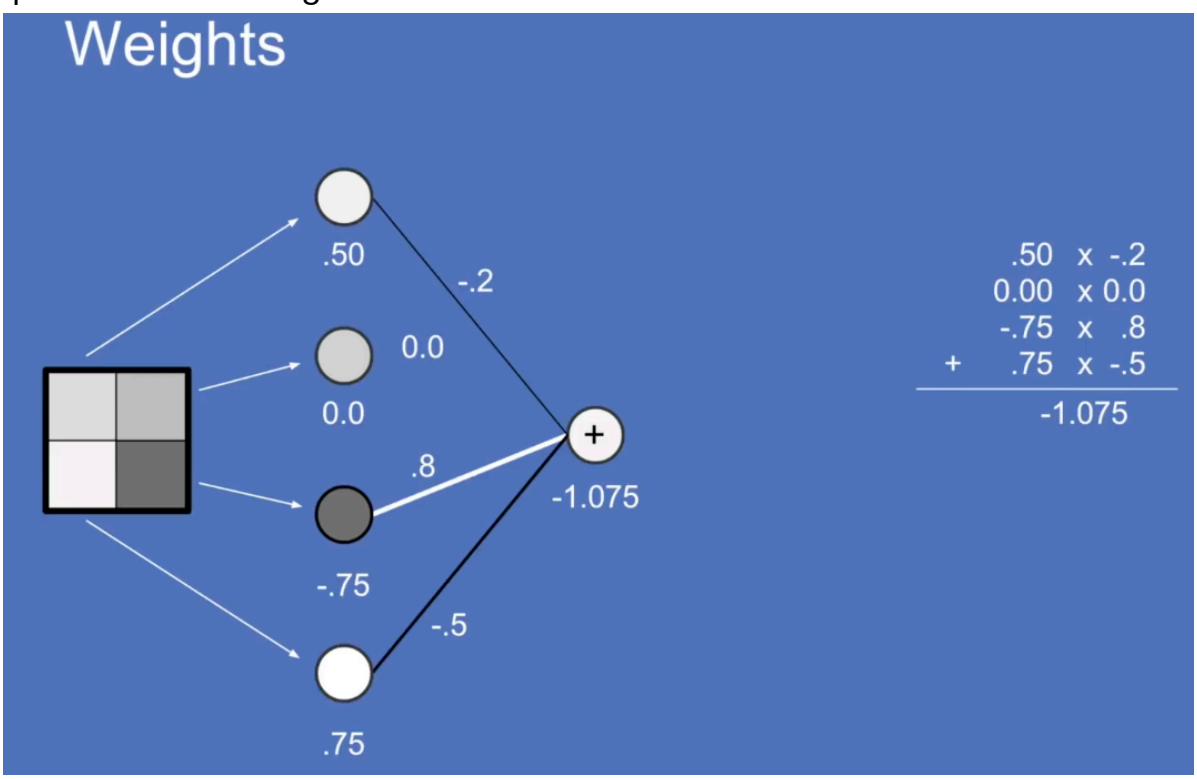
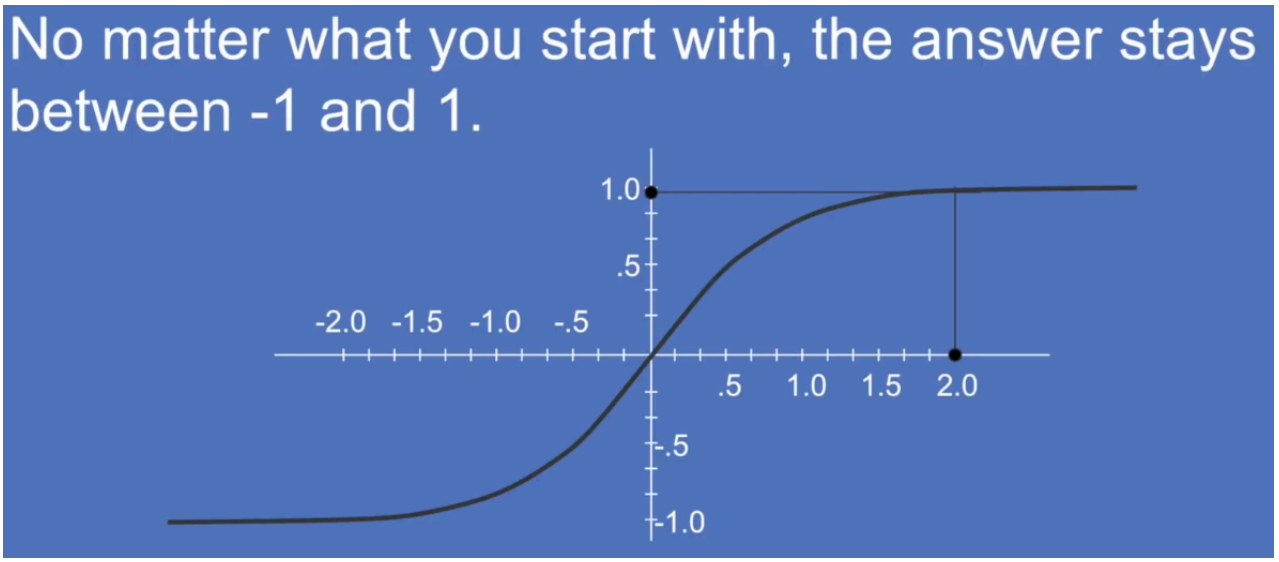
- -> learning types of patterns
- ***-> four pixel camera example***
  - only black and white
  - taking a 2x2 matrix with each of the cells either black, white or grey
  - -> and then wanting to extract whether the line was
    - solid
    - vertical
    - diagonal
    - horizontal
  - ***-> input neurones***
    - -> these are the values of the four cells in the matrix
    - -> each of the pixels has a number from -1 to +1
    - -> the values which the matrix stores can be represented by an array -> the input vector
    - -> the value of each element in the 2x2 matrix is represented by a node in a neural network
    - -> the receptive field of a neurone
      - -> the set of inputs which make the value of the neurone as high as it can be
    - ***-> to build a neural network***
      - -> add up all of the values of the neural network
      - -> each of the connections are weighted
      - -> weight and add the values of the input neurones
      - -> white links are positive, black are negative and the thickness of the line is proportional to the magnitude to the result

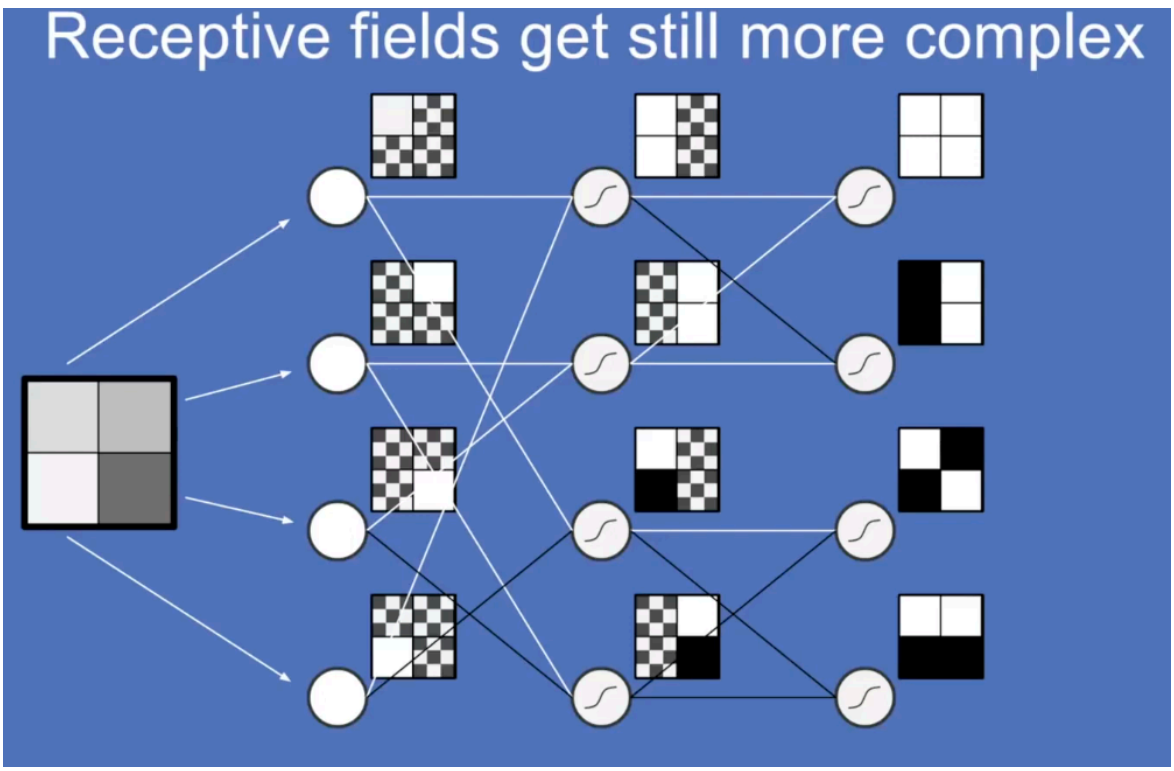


      - -> we are trying to determine if the line is horizontal / vertical et al
      - -> each of the pixels is assigned a number and put into an array (the input vector)
        - -> these (cells in the matrix) are called input neurones
        - -> you add up all of those values
        - -> they each have weights -> you times the value of the input neurone by its weight and do this for all of them in a sum
        - -> after you do this you squash them using a squashing function (a sigmoid)

- ***-> using a sigmoid squashing function***
  - -> adding the weighted input functions
  - -> then using a sigmoid squashing function
  - -> you are inputting the x values and the y is returned
  - -> the output number gets larger but more slowly

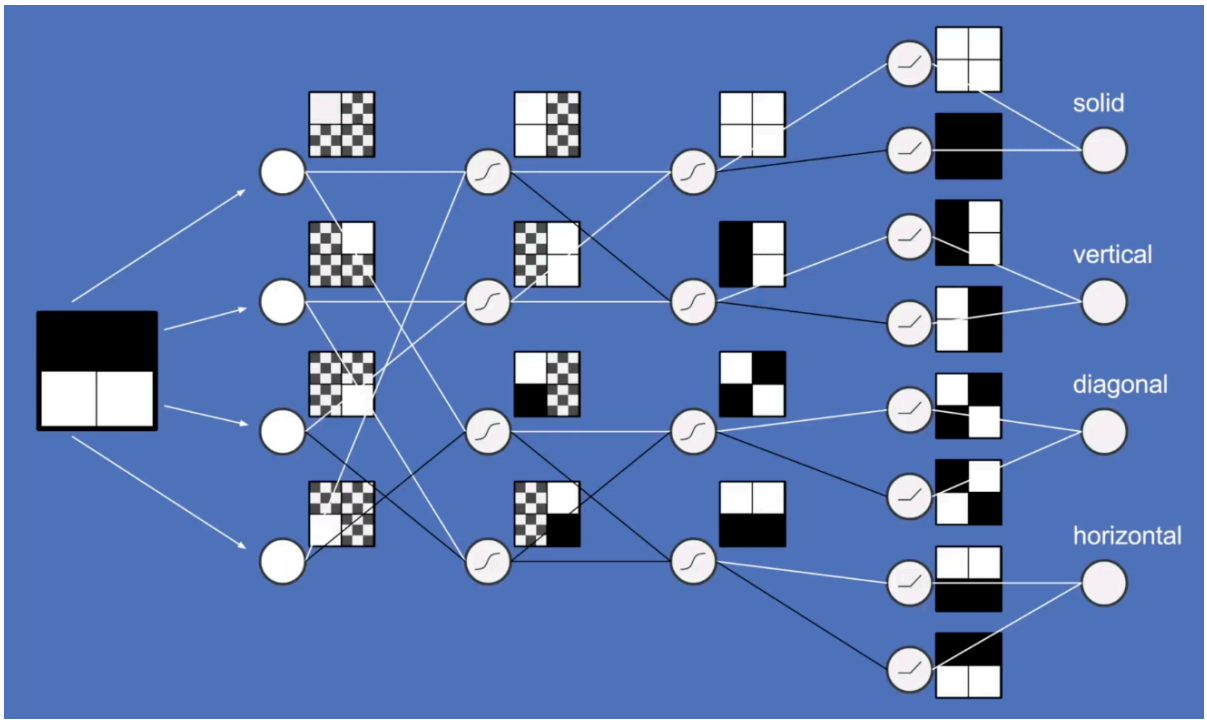


  - -> the range is between ±1 <- the value of the neurones
  - -> you convert the values of the entire matrix into one value using the sigmoid function
    - -> these are the weights which they have
    - -> these weights are processed in multiple layers in the model <- the second with the first layer having been passed through a sigmoid function
    - -> it's covering different combinations of pixel values which the matrix could have <- this gives the receptive field of the neurone
    - -> the output of one layer is the input of the other



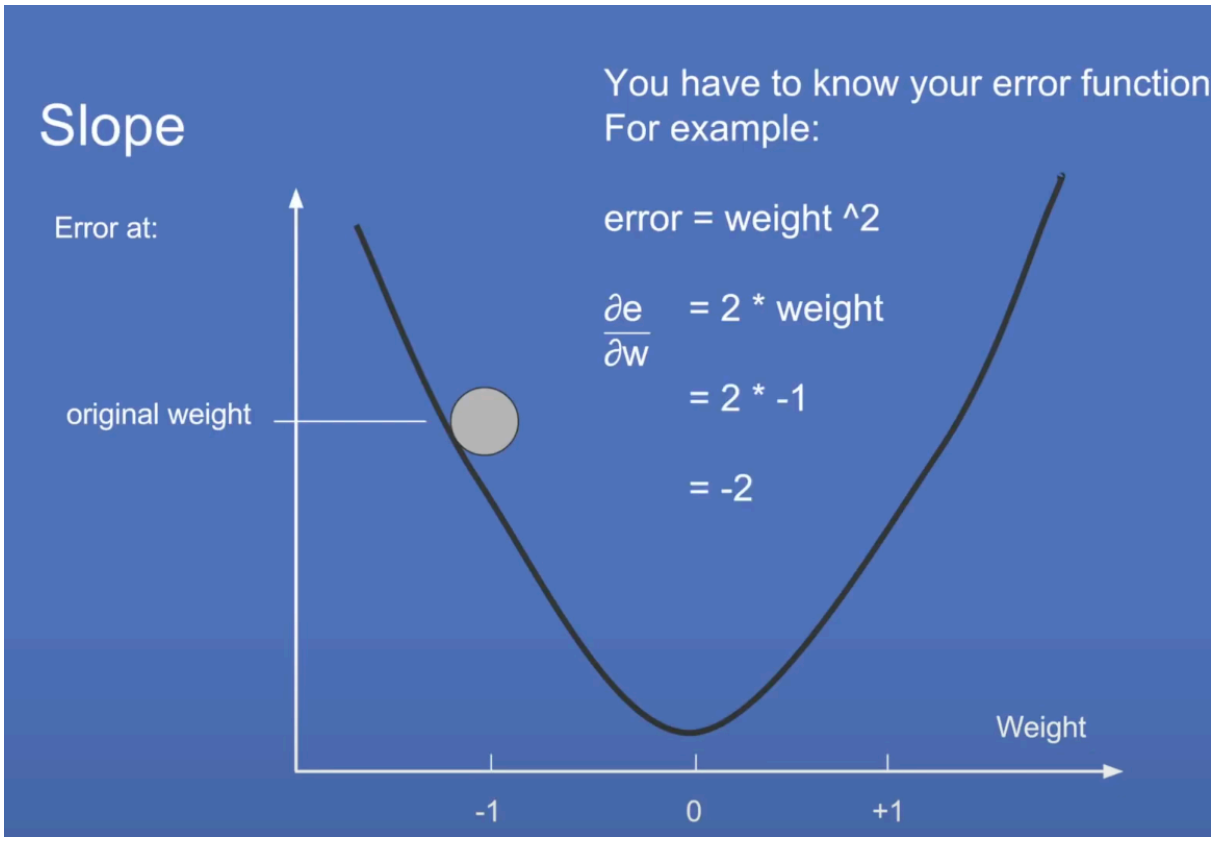
- ***-> another method of doing this -> rectified linear units***
  - -> rectified linear units -> ReLUs
  - -> this is a different type of neurone -> not using a sigmoid squashing function
  - -> this is being used in a layer instead of the squashing function

        - -> you rectify it instead of squashing it
        - -> getting rid of all of the negative values, making them 0
        - -> this can get rid of receptive fields and their opposites
        - -> in this example this is done in the last layer of the network
    - ***-> then we create an output layer***

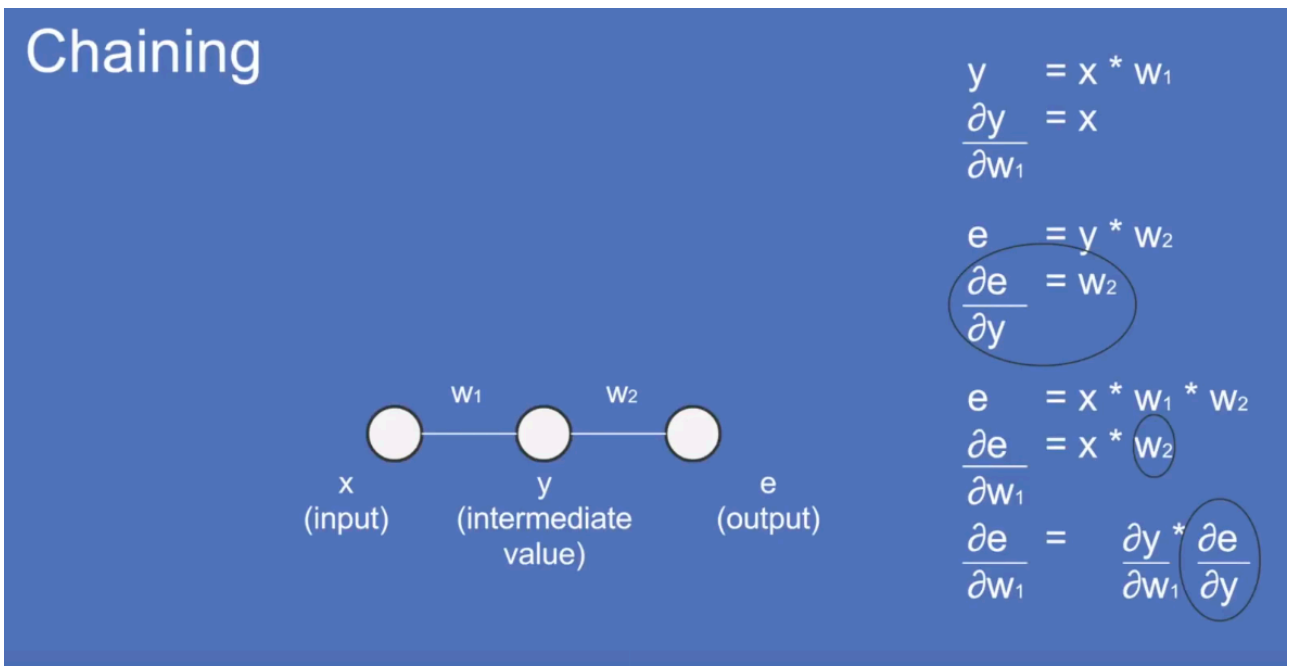


        - -> we are taking the values of the pixels and making them into nodes
        - -> and then adding them together, and doing this with different weights
        - -> then multiplying them together in another layer
        - -> the neural network is outputting that the input matrix was a horizontal line
        - -> each of the output nodes represents the different possibilities for the matrix of pixels - in order to classify it
        - -> the second layer of pixels is for the receptive field of that element
        - -> he is going through he different layers of the neural network and inferring the output of each of the calculations based off of the weights at each stage

- ***errors***
    - ***looking for errors***
        - -> the network is trying to classify the different shape which the four pixcels are in -> it can classify it as the wrong thing
        - -> we want to adjust the weights to minimise the error
        - -> after iterating through the neural network, we can calculate the errors in the predictions it makes
            - -> and then adjust its weights
    - ***gradient descent***
        - -> it's expensive to recalculate the entire neural network every time the errors are calculated
        - -> there is an energy landscape - and it has a lot of dimensions so it's computationally expensive
        - -> calculating the rate of change of error with respect to the weights in the model
            - -> you change the initialisation parameters by x amount, how much does the error in the model's predictions change?
            - -> this is gradient descent
            - -> you are trying to move it into a local / global minimum
        - -> you have to know the error function
            - -> e.g if it's the square of the weight -> you are calculating the gradient of it

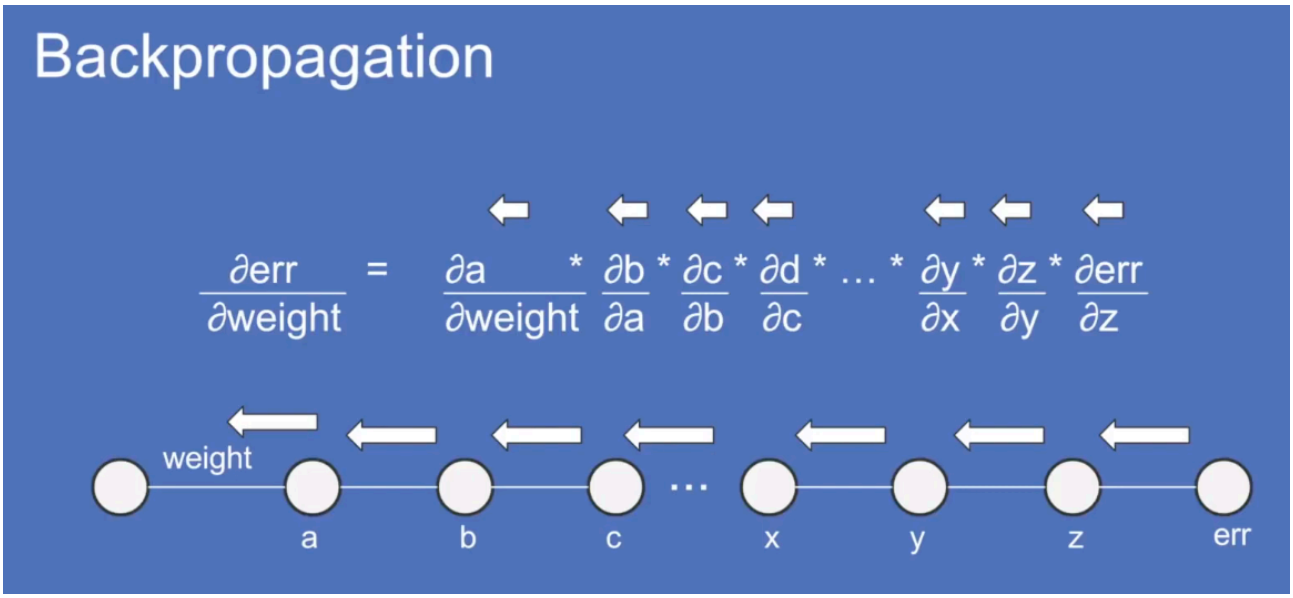


- ***chaining***
  - -> this is for deep neural networks
  - -> consider the simple case with two layers of neural networks
    - -> a hidden layer, a weight, an input and output layer



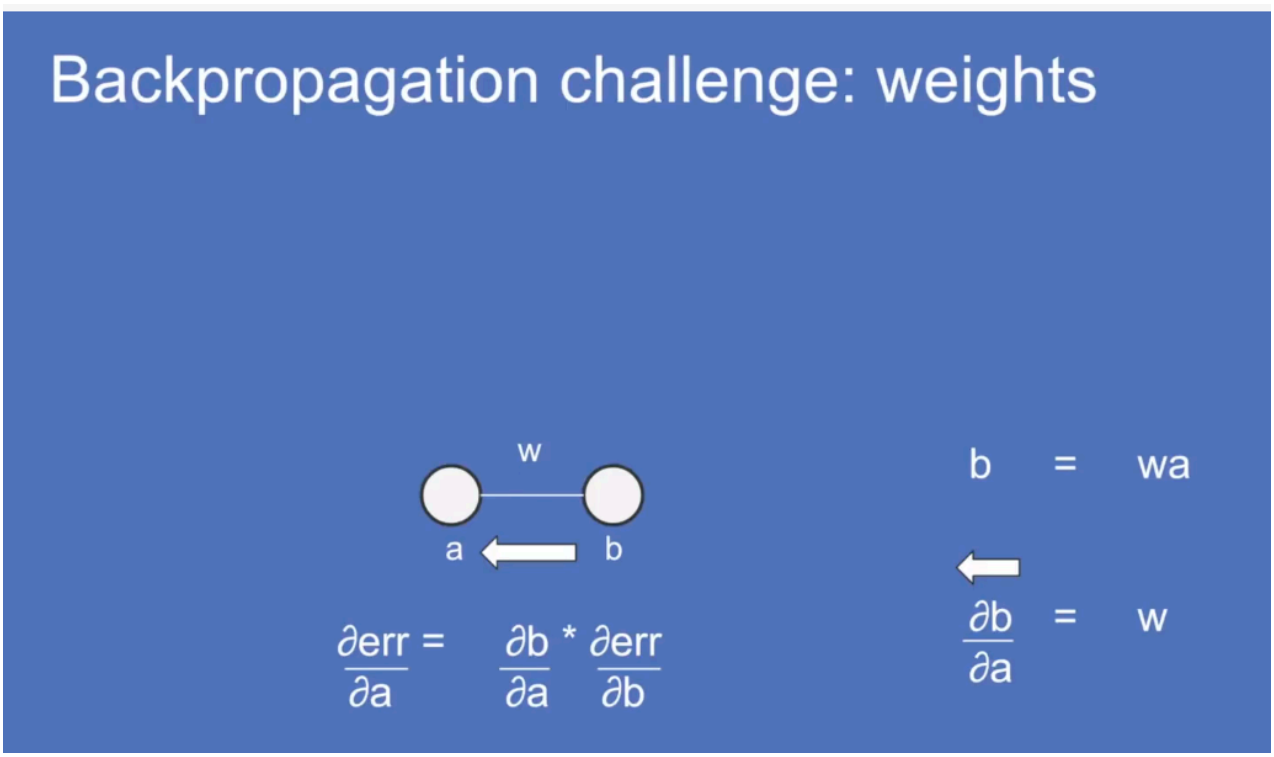
- -> this is a mathematical diagram - similarly to a Feynman diagram -> each layer in it is a layer in a neural network and is represented by maths
- -> w_1 is the weight (probability) that the input x outputs y
- -> so y is w_1x
- -> then because it's too expensive to calculate the weights for the entire model after the error of the predictions it makes is calculated using those weights -> we use that equation to calculate derivatives
- -> for each link in the chain one of these equations can be derived
- -> we take those and take the partial differentials
- -> then we 'chain' them together to get the rate of change of the output with the

initial weight w_1 in the chain

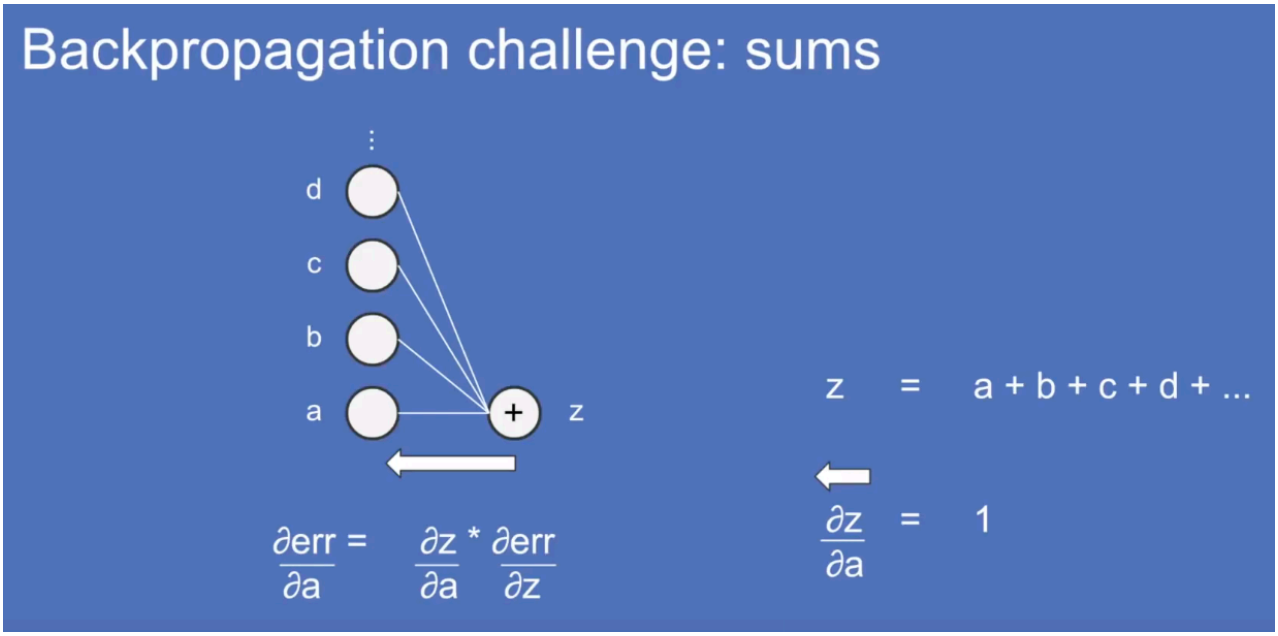


- -> this is that process generalised to a more complicated chain
- -> this is called backpropagation -> because we are calculating the rate of change of the output / error - with the weight of the model at the beginning of the chain by combining these partial differentials
- -> the aim is to choose the value of the weights which minimises the error of the predictions the model makes (we are working in an energy landscape)

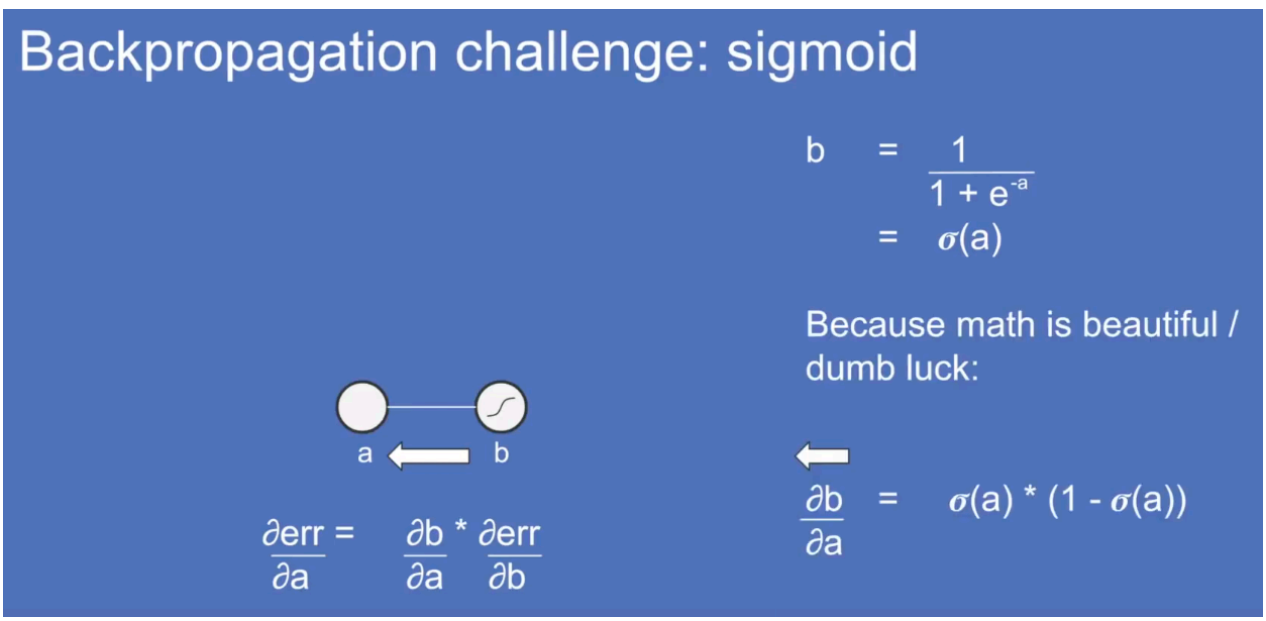
- ***types of backpropagation***
  - -> aka different ways you can rearrange those equations
  - -> they're using the chain rule on the first two nodes
  - -> b = wa -> then turning it into a partial



- ***-> when there are multiple neurones leading into one***
  - -> you can take all of those neurones leading into z and model them as one effective neurone leading into it
  - -> the sum of d(weight) + c(weight) +.... = z <- then you take the partial of the entire thing
  - -> it's probabilistic -> and works like the maths for Feynman diagrams

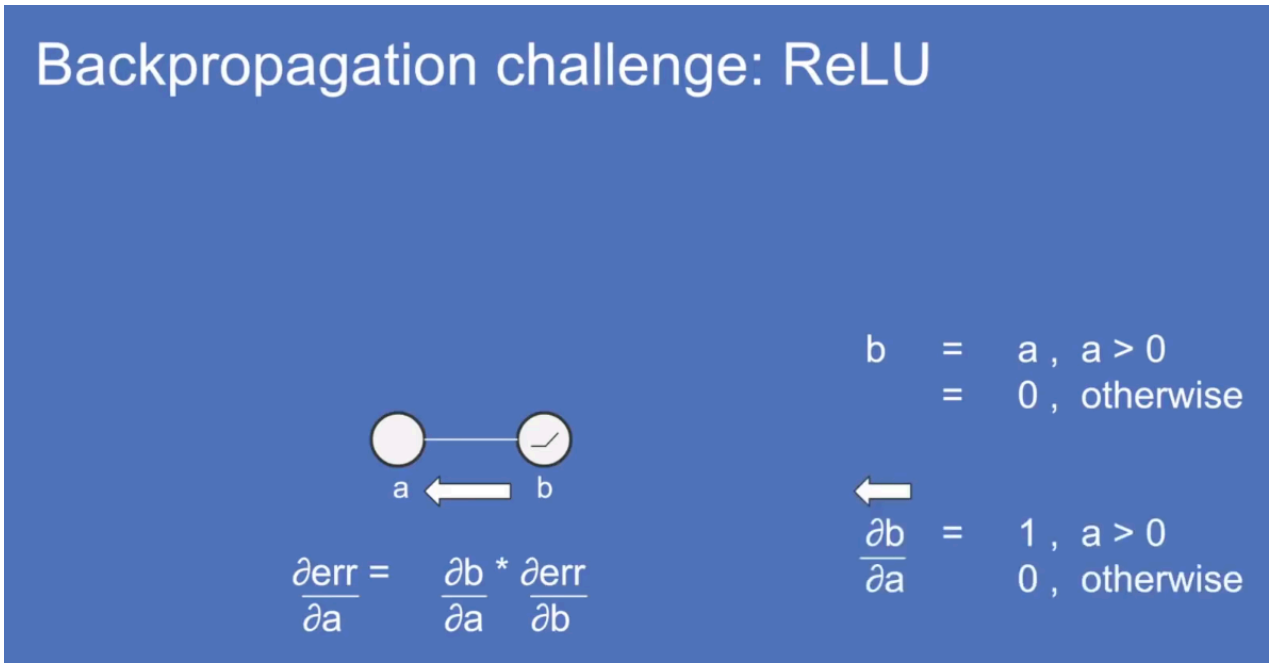


- ***-> back propagating a sigmoid function***
  - -> the equation of a sigmoid function (see image)
  - -> <u>take the partial differential of a sigmoid (squashing) function and you get a function of the same thing</u>

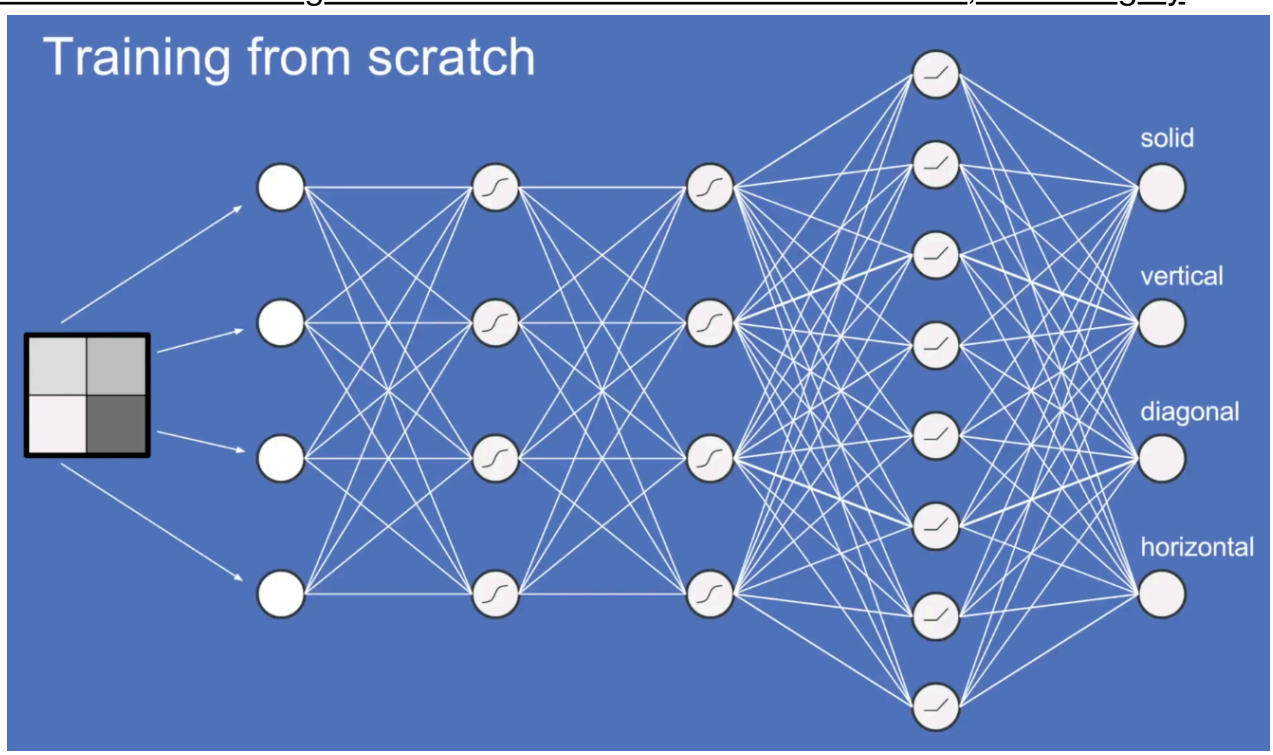


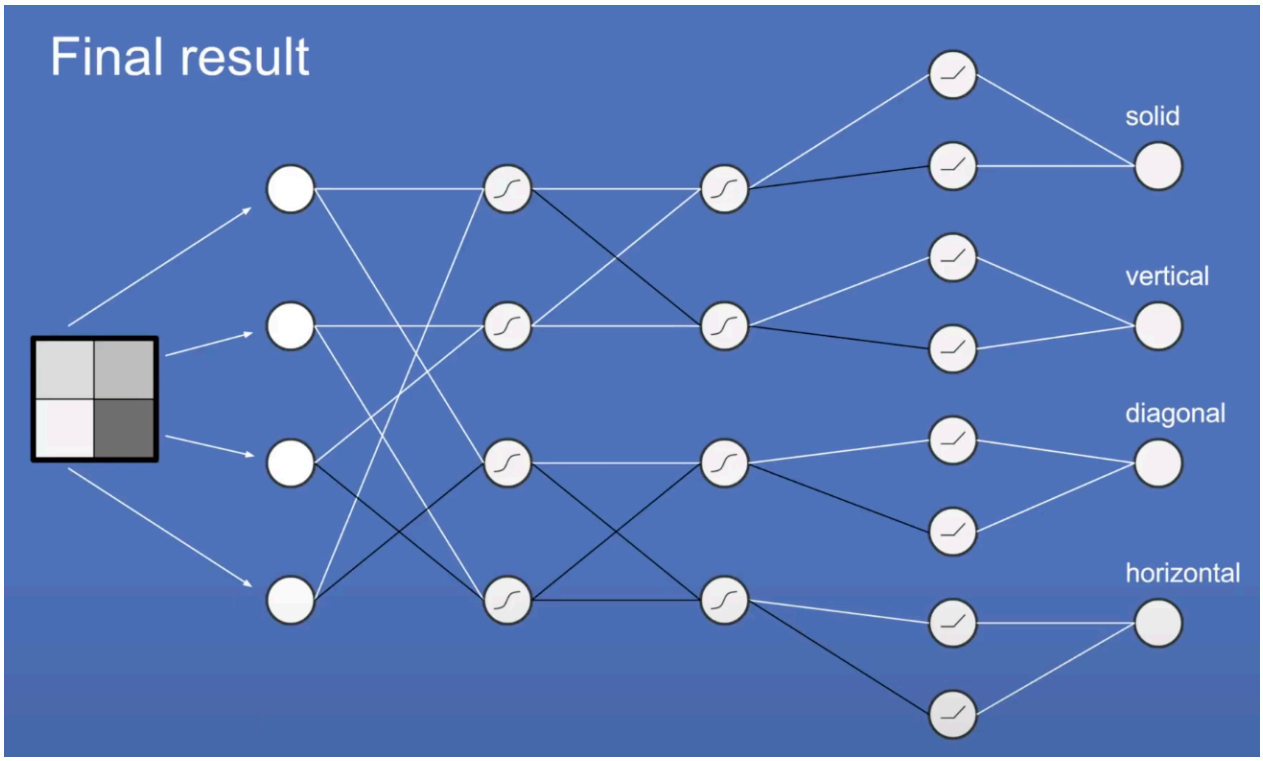
  - -> you times it by 1- itself

- ***-> back propagating an ReLU***
  - <u>-> in other words the same function but with the negatives made zero</u>
  - -> you just take the partial of it if it's positive and the partial of it is negative if it's zero



- ***training from scratch***
  - -> to train, the model starts with a fully connected network and random weights
  - -> we are training the model - so we know what the truth should be
  - -> then we calculate the error based off of the random weights
  - -> calculate the error and adjust the weights
  - -> we carry this on thousands / hundreds of times - and each time use a different input
  - -> then the weights tend to a minimum in the energy landscape -> the rate of change of the weights with respect to the error of the predictions of the model
  - -> each of the output nodes in that model represents a different way the pixels on the matrix could be organised -> and each of them is either black, white or grey





- ***advanced topics***
  - building a neural network
  - -> Karpathy
  - -> deep learning is another example
  - -> there are also different blog articles which he's recommending (see images)

  - -> is it better to calculate the gradient (slope) directly rather than numerically since it is computationally expensive to go back through the entire neural network and adjust the weights for each layer of the neural network

Data Science and Robots Blog

For more How it Works:

- How Deep Learning works
- How Convolutional Neural Networks work
- How Bayes Law works
- How data science works
- How linear regression works

These slides

https://docs.google.com/presentation/d/1AAEFCgC0Ja7QEI3-wmuvIizbvaE-aQRksc7-W8LR2GY/edit?usp=sharing