Cs229_L-11_Nueral_Networks

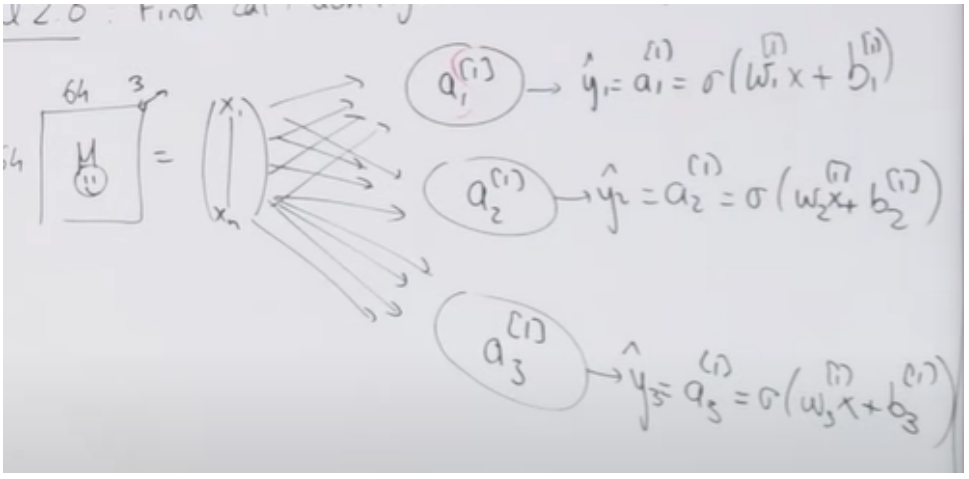
Logistic regression

Find a needle image in a haystack image:

- Take the w*l*b bit image and convert it into a linear vector X with w*l*b,1 dimension
- $use\ y = sigma(wx + b) = \sigma(\theta^T X)$
- w has the dimension 1,w*l*b
- To optimize w and b we use logistic loss $L = ylogy' + (1 - y)log(1 - y')$

Logistic regression → Nueron

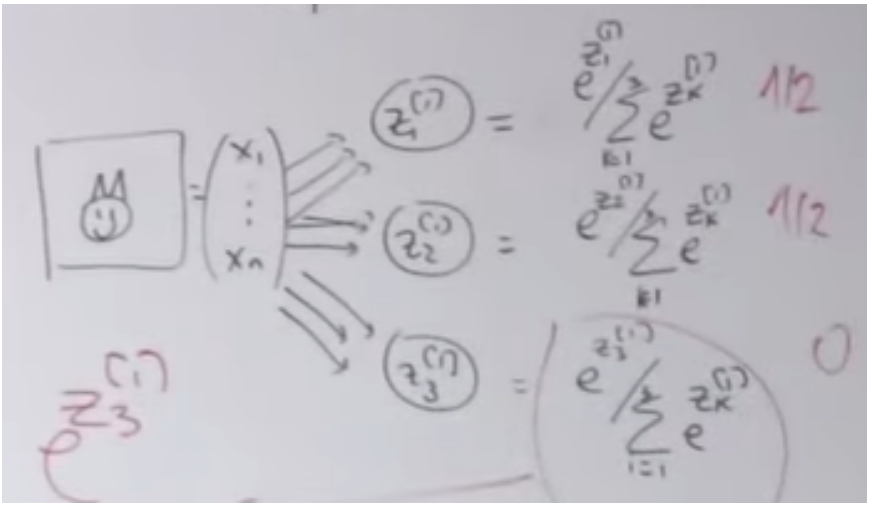
1)Detect three different types of image in a haystack image



- after maping image into a vector send it to three different logistic regressors.
- y will be a 3*1 vector instead of a singular number.

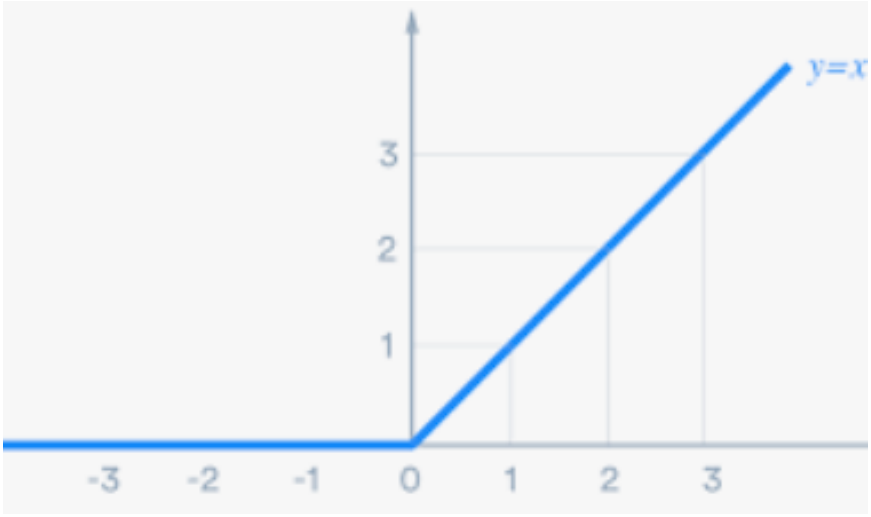
- The three sigmoids are independent preferable for multiple detection than classification.
- Loss is sum of all three losses
- When you differentiate you only take partial leading to independent value of each w and b.

2)Detect 1 needle image out of three possible needle image in haystack image



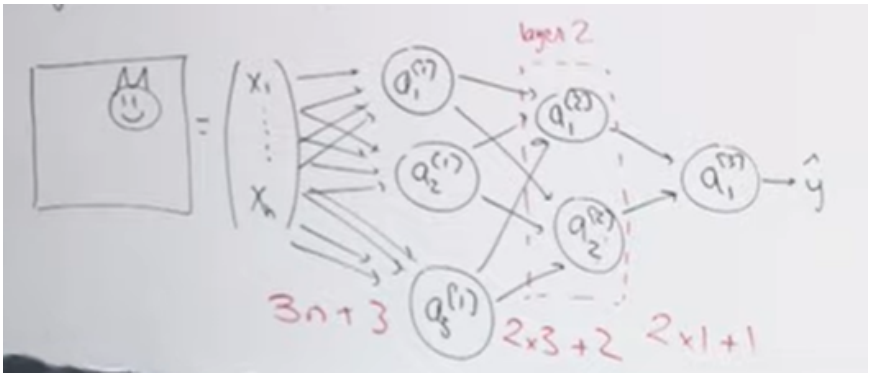
- we get z^i as $w^i x + b^i$ vector we do $\frac{e^(y^i)}{\sum e^(y^i)}$
- basically we do softmax regression-soften probabilities to have a sumf 1 and find max
- $L = \sum y_k log y'_k$

Regression → Nueron



- suppose we want to get an output other than 0 and 1.We can make a nueral network based on linear regression
- instead of using $y = \sigma(wx + b)$ we can use $y = relu(wx + b)$
- L1 loss and L2 loss are loss functions

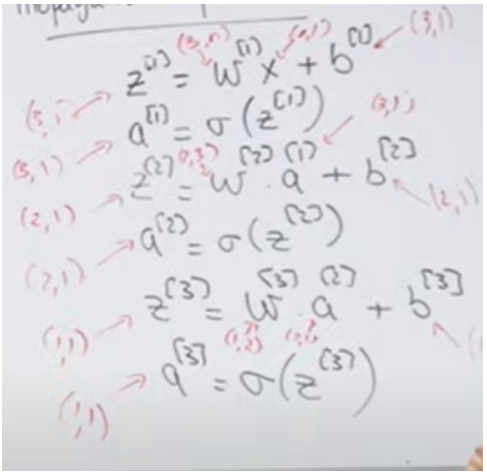
Nueral Networks



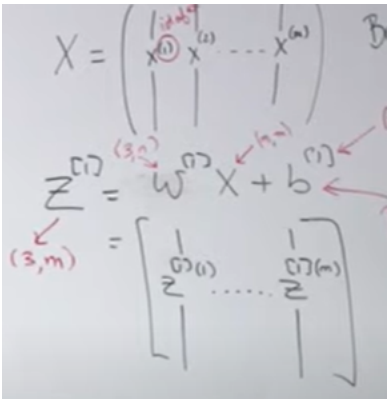
- we will let the nueral network figure out what are the best inputs to be forwarded to next layer,

$$z^i = w^i x + b^i$$
$$a^i = \sigma(z^i)$$

- The dimensions of z w b a change based on the layer they are in.a has same dimension has z.

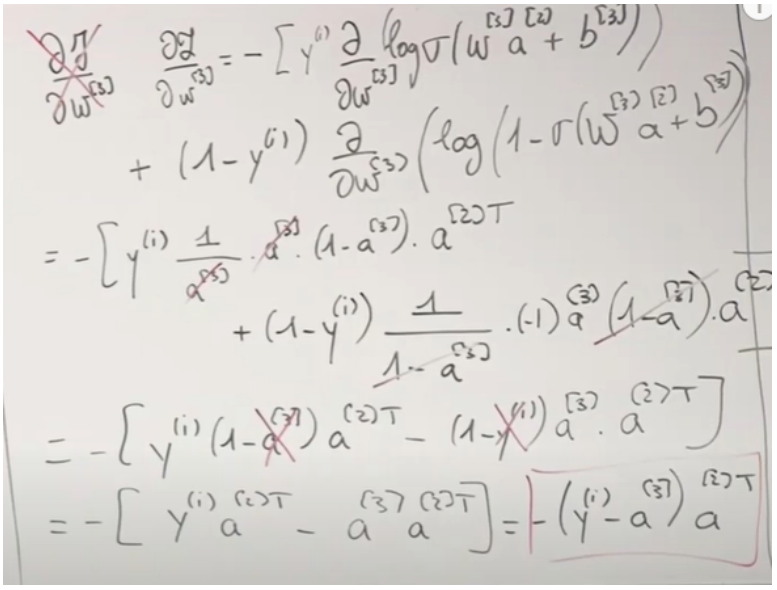


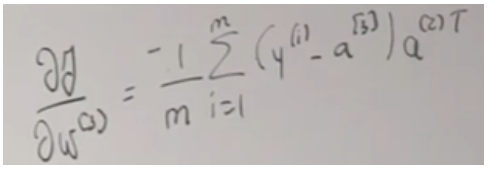
- here we are taking only one image at a time. Hence it is a stochastic gradient descent
- For Batch gradient descent we just add all images as a multiple columns of a matric. Broadcasting -parallize code without increasing the number of parameters



a) Back propagation

- what we mean by back propagation is back-propagation of weights
- after initializing random values in all **w** and **b** parameters we would need to update **w and b** based on $\frac{dJ}{dw^i}$ and $\frac{dJ}{db^i}$
- We will first find $\frac{dJ}{dw^3}$ and then store some of the intermediate steps in cache and then used them to find $\frac{dJ}{dw^2}$ and $\frac{dJ}{dw^1}$
- This occurs because we move through the propagation equation in reverse when we are applying chain rule to find each differential





- $\frac{dJ}{dw^3} = \frac{dJ}{da^3}\frac{da^3}{dz^3}\frac{dz^3}{dw^3}$ where $\frac{dz^3}{dw^3} = a^{2T}$

- $\frac{dJ}{dw^2} = \frac{dJ}{da^3}\frac{da^3}{dz^3}\frac{dz^3}{da^2}\frac{da^2}{dz^2}\frac{dz^2}{dw^2}$

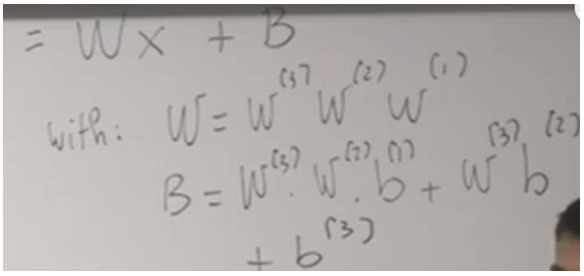
1. we can get $\frac{dJ}{da^3}\frac{da^3}{dz^3} = a^3 - y^i$ from the previous equation by storing it in cache
2. $\frac{dz^3}{da^2} = w^{3T}$ so the value of this partial is dependent on the value of weight of next layer.
3. $\frac{da^2}{dz^2} = a^2(1 - a^2)$ so the value of this partial depends on the value of activation function of this layer stored in cache.
4. $\frac{dz^2}{dw^2} = a^{1T}$ so value of this partial depends on value of activation function of previous layer stored in cache

5.All values required during backward propagation is stored in cache during forward propagation

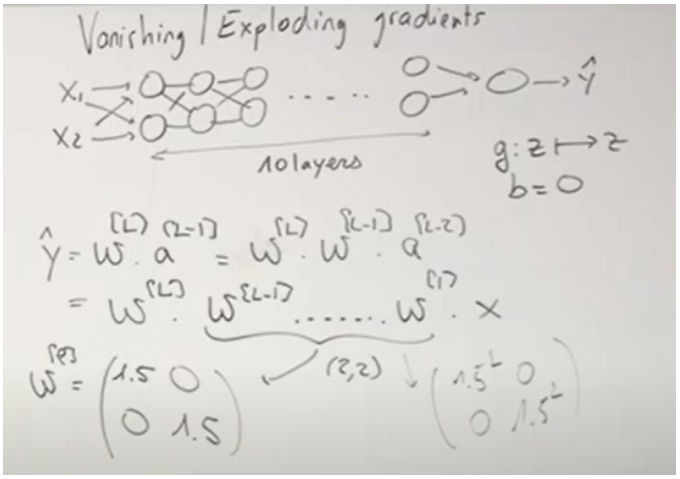
6.we know $\frac{dJ}{dw^2}$ $depends$ $\frac{da^3}{dz^3}$ but in sigmoid function for large values of z this gradient tends to 0.Hence sigmoid works well only in the non saturating part.

b) Activation function

- Lets say we dont use activation function and keep on passing z to the next layer the our propogation equations will colapse to
- $z^1 = w^1 x + b^1$
  $z^2 = w^2 z^1 + b^2$
  $y = z^3 = w^3 z^2 + b^3$

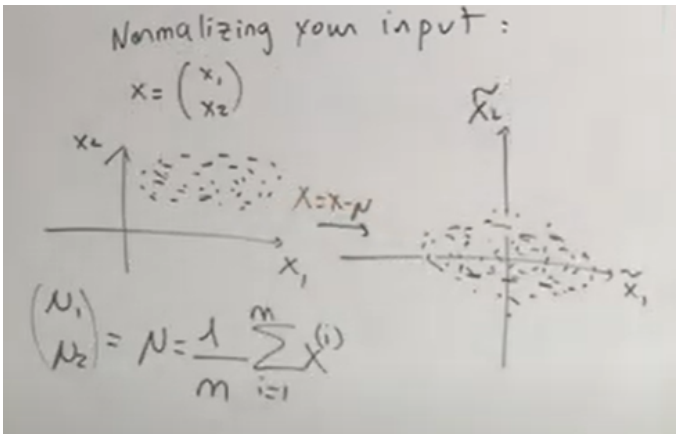


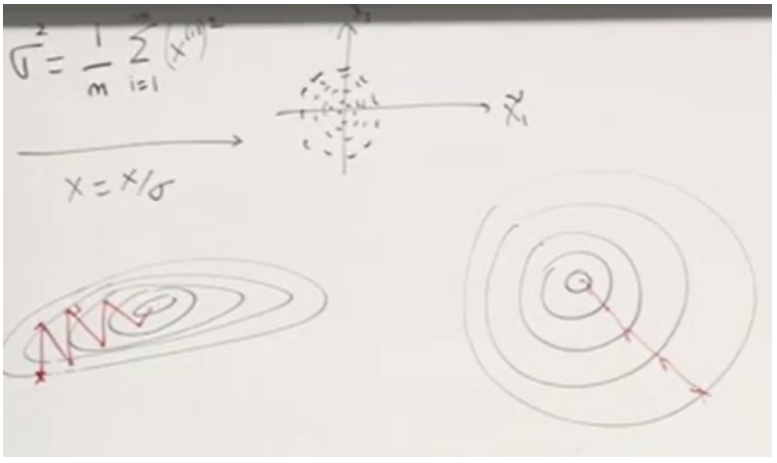
- activation functions add non linearity in the network otherwise our end network will just be a linear regressor.
- activations function also help in thresholding
- Absence of activation functions also causes Vanishing or Exploding gradient



c) Initialization methods

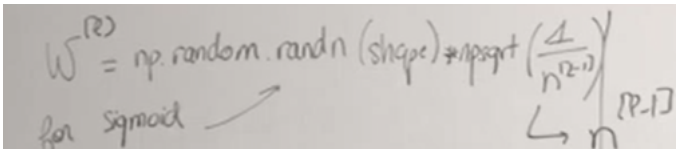
1) We know large values of z can lead to saturation especially in the first layer. So we need to normalize our data

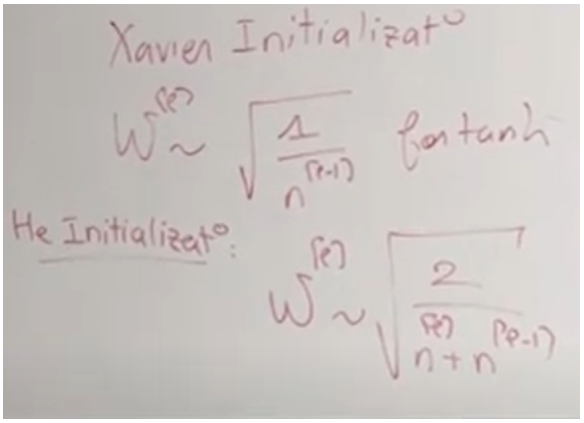




2) To avoid vanishing and exploding gradients we initialize all weights to approx 1/n to keep z within a particular range

- Standard initialization-sigmoid

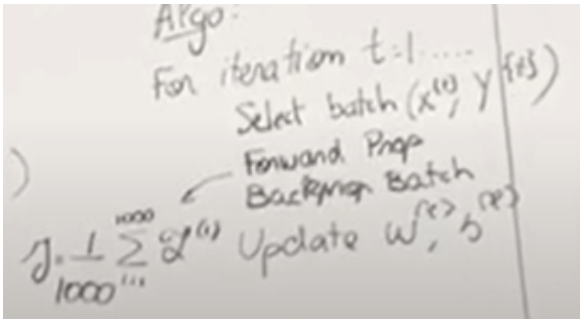




3)Just because we want all the weights to be approximately equal to some value doesnt mean we can have all weights to be initialized to same value. If all weights in a layer are initialized to same value they would end up learning the same feature leading to a symmetric network.

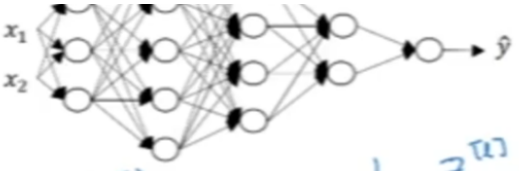
d)Optimization

- Regular gradient uses vectorization but this leads to longer computation times ,stochastic gradient takes 1 instance at time but the weights never converge to a stable value .
- We use mini batch gradient descent as



e)Broadcasting

- In broadcasting we give parallel inputs(parallel columns) instead of a single column



$z^{[1]} = W^{[1]} x + b^{[1]}$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, 1) \quad (n^{[1]}, 1)$

$[z^{1(1)} z^{1(2)} \cdots z^{1(m)}]$

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$(n^{[1]}, m) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, m) \quad (n^{[1]}, 1) \rightarrow (n^{[1]}, m)$

$z^{[l]}, a^{[l]} : (n^{[l]}, 1)$

$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$l=0 \quad A^{[0]} = X = (n^{[0]}, m)$

$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m$

Andrew Ng