DEEP LEARNING

PAGE No.
DATE / /

→ Uses neural network
* Convolutional neural network → good for image recognition
* Long Short-term memory network → good for speech recognition

Neural networks are inspired by brain

Neuron → a thing that holds a number

First layer
→ One neuron corresponds to each pixel and holds grayscale value for pixel (0 - black to white 1) This no is called Activation

→ Last layer has one neuron for each possible output
→ Hidden layers btw 1st & last layer
→ Pattern is generated in each layer when image is fed to the network
→ The brightest neuron in last layer is the input.

Eg - If we want to recognize a number
→ The second last layer

can detect loops and lines that make up the number
→ The layers before that can detect individual edges of loops and lines

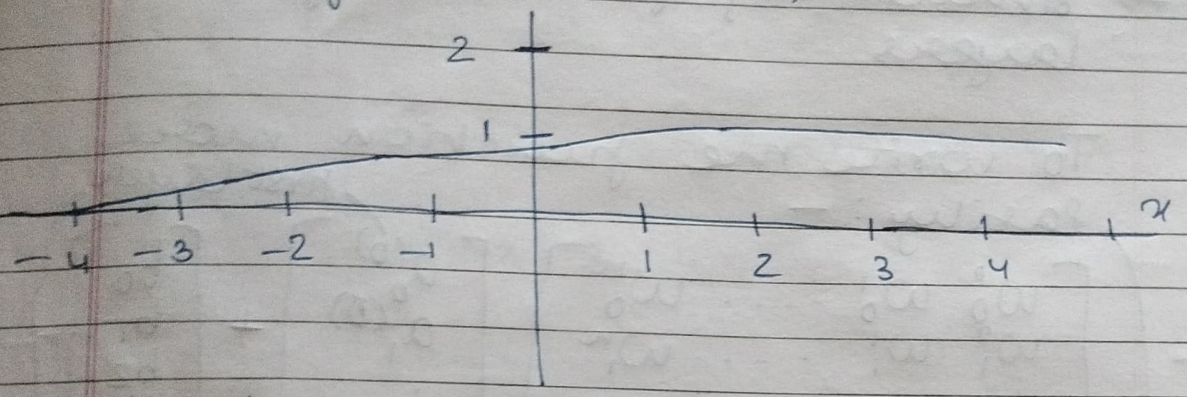
network shld convert:
pixels → edges → loops & lines → no

* What parameter should network have to recognize an edge in a region :—
→ Assign weight to connection between neurons of 1st & ~~2nd~~ layers ~~called~~
→ take all activations of first layer and compute weighted sum as:—
$a_1 w_1 + \dots a_n w_n$
→ This sum will give positive value for region containing a component of the loop
→ For edge detection ⟹ negative weights are associated with surrounding pixels as they are darker
→ Since we need values btw 0 & 1 we then pump the weighted sum into a function to give a value btw 0 & 1

PAGE No.
DATE / /

→ A common function that does this is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

(logistic curve)



∴ Activation of neuron
$= \sigma(a_1w_1 + a_2w_2 \cdots a_nw_n)$

Suppose we dont want to activate neuron & meaning fully for weighted sum greater than 0 but for some other limit such as sum > 10

∴ We need bias for inactivity so we subtract some no from weighted sum before applying sigmoid func

Eg $\sigma(a_1w_1 + a_2w_2 \cdots a_nw_n - \boxed{10})$

↓ sigmoid ↑ How positive is this? ↓ "bias"

→ All neurons of 2nd layer are connected to neurons

of 2nd layer.
→ Each neuron in second layer will have some bias associated with it
→ Similarly for all following layers

To write the function more easily :-

$$\sigma\left(\begin{bmatrix} w_0^0 & w_0^1 & \cdots & w_0^n \\ w_1^0 & w_1^1 & \cdots & w_1^n \\ \vdots & & & \\ w_k^0 & w_k^1 & \cdots & w_k^n \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}\right)$$

(first layer)

$$= \begin{bmatrix} \sigma(a_0^0 w_0^0 \cdots a_n^a w_0^n - b_0) \\ \vdots \end{bmatrix}$$

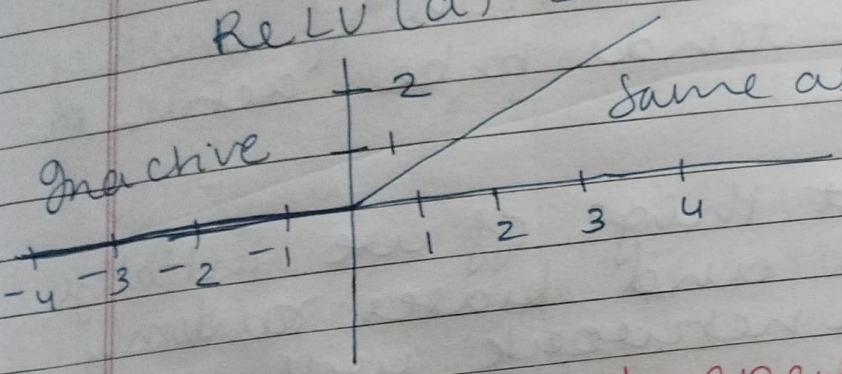
⇒ What this represents is :

$$\sigma(\underbrace{w}_{\text{weight matrix}} \underbrace{a^{(0)}}_{\text{activation matrix}} + \underbrace{b}_{\text{bias matrix}}) = a^{(1)} \text{ activation of neuron in 2nd layer}$$

Each neuron can be thought of as a function that takes outputs of all neurons in prev. layer and gives a no btw 0 & 1

⇒ Modern networks dont use sigmoid anymore they use ReLU (Rectified linear unit) as its much easier to train

ReLU(a) = max(0, a)



How the network learns?

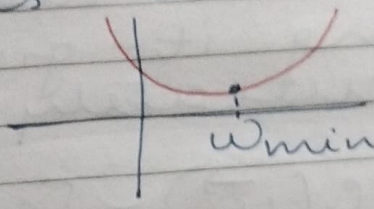
⇒ What we want is an algorithm where u can show this network a whole bunch of training data in form of images with labels showing what they are and the machine will adjust its weights and biases accordingly based on training data which can be generalised to images beyond training data.

⇒ Machine learning is like a calculus exercise and basically comes down to finding the minimum of a function

→ Each neuron of a layer is connected to every neuron in prev layer & weights are the strength of these connections
→ Bias is indication of whether neuron tends to be active or inactive

→ To start things off we initialize weights and biases randomly
→ The nn network will perform horribly and last neuron layer will be a mess
→ So we define a cost function To do this we add up squares of differences of the output activations and the value that we want them to have which called the cost
→ Cost is small when network is more accurate
→ For each image ~~for each neuron~~ there will be a cost
→ We find the average cost for all the images (~~for each neuron of output layer~~)
(cost is found for each output neuron and then added up)

⇒ rather than taking input of all the weights of each layer as input we use a single input :- $C(w)$ which gives a minimum output



$$\frac{dC(w)}{dw} = 0$$

⇒ However minima won't always be feasible (complicated functions)

⇒ So instead start at a random input and which direction to step to lower the output

⇒ However by this method the local minimum we found or may not be the lowest possible value

⇒ Gradient of a function tells us direction for steepest ascent whereas negative of gradient gives steepest descent

⇒ So algorithm for finding w_{min} is

1) compute ∇C (gradient
2) small step in $-\nabla C$ diren
3) repeat step 2

$$\vec{w} = \begin{bmatrix} \\ \end{bmatrix} \leftarrow \text{all wts \& biases} \qquad -\nabla C(\vec{w}) \leftarrow \text{how to nudge the wts \& biases} \begin{bmatrix} \\ \end{bmatrix}$$

$\vec{w} - \Delta C(\vec{w})$ — will decrease cost function

→ cost function is average for all samples so minimizing it gives output more accurately in each sample

Eg: $-\nabla C(\vec{w}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.18 \\ -0.37 \\ 0.16 \end{bmatrix}$

It will take inputs equal to no of weights (ie connections btw neurons)

The sign of this func tells what shld inc/ decrease

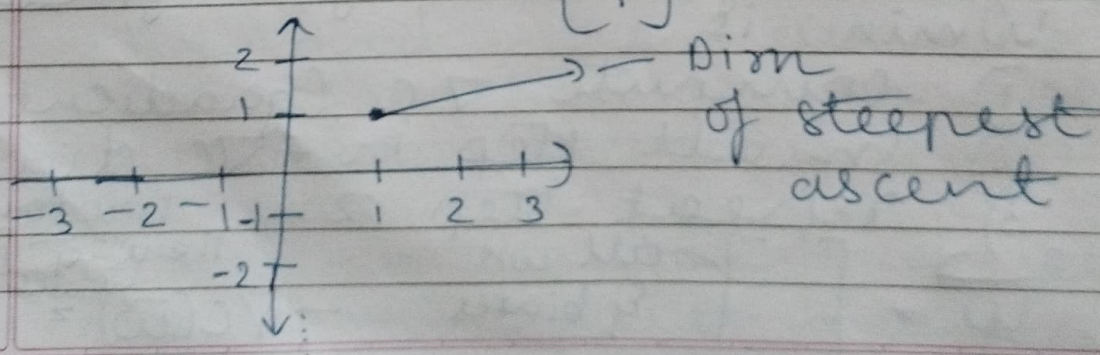
→ The value tells whether it shld decrease a lot or little or somewhere in between

→ Some adjustments matter more than others

Eg Suppose we have a cost function with only 2 inputs

$C(x,y) = \frac{3}{2}x^2 + \frac{1}{2}y^2$

and $\nabla C(1,1) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$



Since $\nabla C(1,1) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ we can say that changes to x have 3 times as much impact as comp. to changes to y

→ Even though image recognizes images it cannot generate them. So if we give a nonsense image it will confidently give a single output which shouldn't happen

Backpropagation

→ Since cost function involves averaging over thousands of training examples the way the Weights and biases for each step are adjusted depends on each training example

→ Suppose we focus on one training example only. If the network is still working badly we will get a garbage output It is important to keep track of which values changing are important (eg if a wrong output is close to 0.0 we ignore it and focus on the outputs closer to 1 which need to be decreased and also focus on correct o/p that must be increased)

Suppose we need to increase activation of correct output whose activation is given by $\sigma(w_0 a_0 \cdots w_{n} a_{n} + b)$

We can do this in 3 ways

→ Increase b

→ Increase w_i — in prop to a_i

→ change a_i — in prop to w_i

* If we only concentrate on changing weights (w_i) we see that different weights have differing level of influence

* Connections from brighter neurons of previous layer have more influence since they are multiplied by larger activation values ($w_i a_i$)

→ Hence there is a strong connection between neurons of layer that are more active and neurons of next layer that we wish to become more active

* Another way is to increase activation (a_i) of everything connected to ideal o/p neuron with +ve weight became more active and with −ve weight became less active.

Most effective when seeking changes proportional to

size of corresponding weights (w_i)
We cannot change activation we can only change weights & biases (since activation depends on brightness of pixels) its helpful to keep a note of what desired changes are

⇒ This is only for the output neuron and what it wants

⇒ We also have to decrease activation of other neurons which have their own desires as for what shld happen for their activation to decrease

⇒ All these desires of each output neuron as to what shld happen to 2nd last layer is added together in proportion to w_i and in proportion to a_i

⇒ By adding together all these desired changes u get a list of nudges that you want to happen to 2nd last layer

⇒ This is backpropagation

⇒ It shows how each training example wishes to nudge weights & biases

⇒ If we do this for only one image sample the network would only be able to identify that category of images

→ So we do it for all images and average it out
→ Hence the collection obtained of all the nudges to weights & bias ultimately gives the −ve gradient ie

$$-\nabla C(w_1, w_2, \ldots w_{13,001}) = \begin{bmatrix} -0.08 \\ +0.12 \\ -0.06 \\ \vdots \\ 0.04 \end{bmatrix}$$

→ It takes computers extremely long time to add up nudges for all examples and gradient descent steps. So instead:

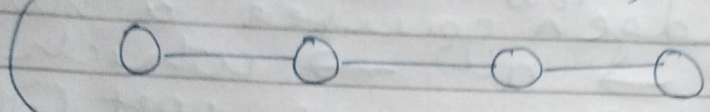
* Training data is shuffled into mini batches
* Then compute cost of a step for mini batch which wont be the actual gradient of cost func.
* Each mini batch gives a good approximation as well a computational speed up

Stochastic gradient descent
↓
use one mini batch for each step
↓
rather than all data for all steps

Backpropagation Calculus

→ If we start with a very simple layer with 3 weights and 3 biases

C (w1, b1, w2, b2, w3, b3)



→ our goal is to understand how sensitive the cost function is to these variables

→ Let activation of last neuron be $a^{(L)}$ & second last as $a^{(L-1)}$

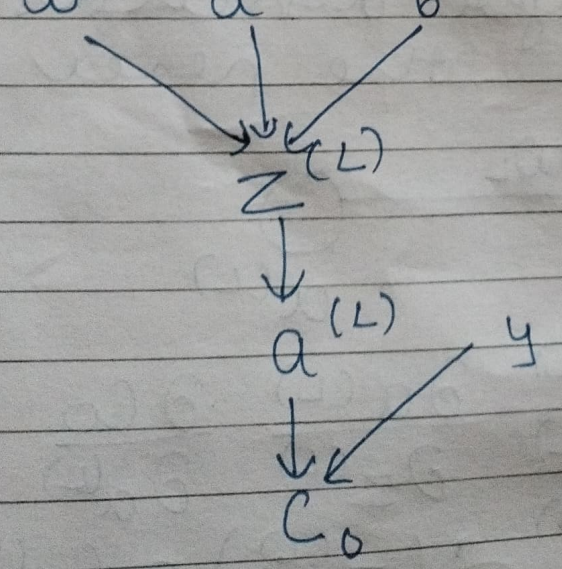
→ Suppose desired value of $a^{(L)}$ is y (Eg 1.0)

→ ∴ Cost of this network for single training eg is

$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



We can imagine a seperate no. line for w, z, a, C_0

→ The goal is to find how sensitive the cost function C_0 is to w

∴ we want $\frac{\partial C_0}{\partial w^{(L)}}$

→ Tiny nudge to $w^{(L)}$ causes a nudge to $z^{(L)} \rightarrow a^{(L)} \rightarrow$ which directly influences cost (∂C_0)

$$\therefore \boxed{\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} * \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial C_0}{\partial a^{(L)}}}$$

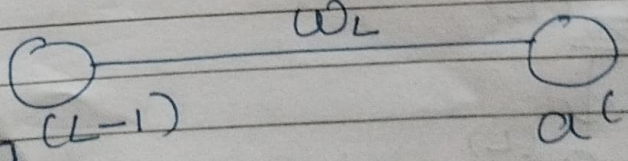
(chain rule)

$$\frac{\partial C_0}{\partial w^{(L)}} = 2(a^{(L)} - y)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\boxed{\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}}$$

The amount that the small nudge of wt influenced the ~~last~~ layer depends on how strong the perev neuron is



$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$= a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$

(Since C is found by averaging cost over all examples, the derivative of C is by averaging derivative of cost over all examples

This is just the component of gerading vector

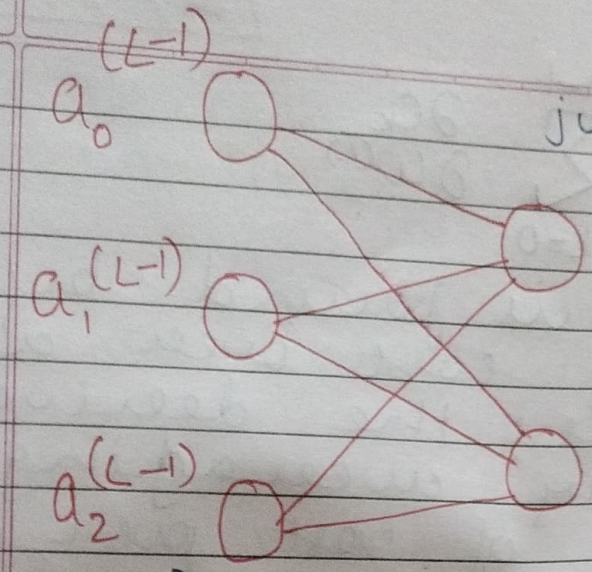
$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

→ Sensitivity Similarly for bias

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = 1 \cdot \sigma'(z^{(L)}) \times 2(a^{(L)} - y)$$

→change for bias remains same almost always

If we do this same thing for a more complicated layer



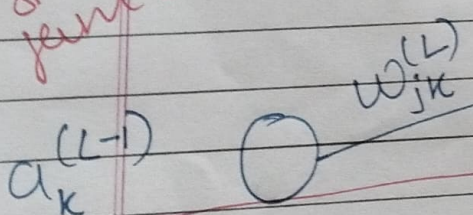
There will just be more indices to keep track off

$(a_k^{(L-1)}$ $(a_j^{(L)})$

Now it will be

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$

since j starts from 0



$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + \cdots + b_j^{(L)}$$

$$a_j^{(L)} = \sigma(z^L)$$

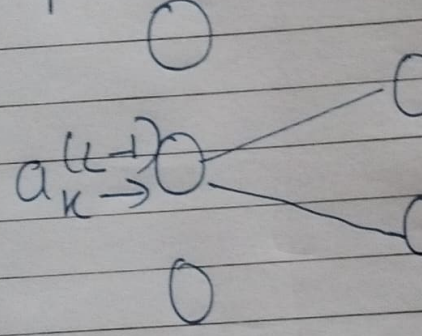
$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L - 1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial C_0}{\partial a_j^{(L)}}$$

sum over

The difference is that the neuron influences cost function through multiple different paths



$$\nabla C \leftarrow \begin{cases} \dfrac{\partial C}{\partial w_{jk}^{(L)}} = a_k^{(L-1)} \sigma'(z_j^{(L)}) \dfrac{\partial C}{\partial a_j^{(L)}} \\ \sum_{j=0}^{n_{L+1}-1} w_{jk}^{(L+1)} \sigma'(z_j^{(L+1)}) \dfrac{\partial C}{\partial a_j^{(L+1)}} \\ \text{or} \\ 2(a_j^{(L)} - y_j) \end{cases}$$