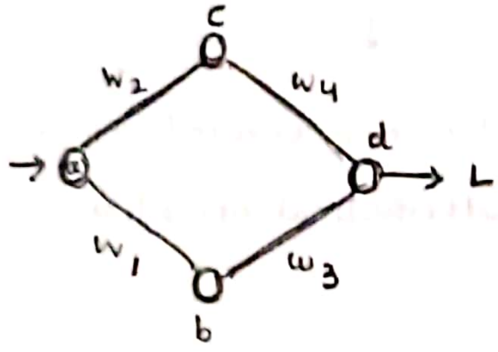
autograd library

*) Backward pass : use of chain rule to compute the gradients of weights wrt. the loss fn.

diagram I



$b = w_1 * a$

$c = w_2 * a$

$d = w_3 * b + w_4 * c$

$L = 10 - d$

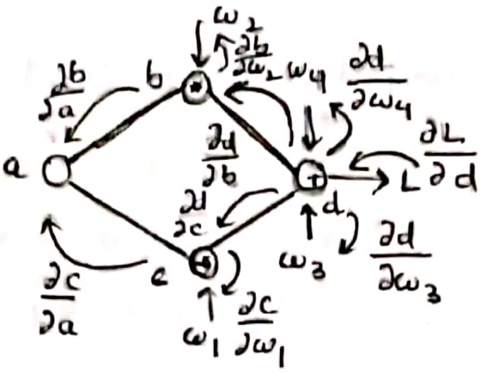


diagram II

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_4}$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_3}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_2} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial c} * \frac{\partial c}{w_2}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_1} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial b} * \frac{\partial b}{\partial w_1}$$

*) All the gradients on RHS of the eqns mentioned above can be computed directly since the numerators of the gradients are explicit functions of the denominators.

*) computation graph : galvanising the idea of somehow being able to seamlessly compute the gradients, regardless of the ~~[illegible]~~ architecture of the network, so that programmers don't need to the computation manually in form of a data structure is called a computation graph.

*) computation graph is very similar to the diagram II. where the nodes are reflective of operators. operators are basic mathematical operators, except the case when the node represents a user defined variable tensor.

(a, w_1, w_2, w_3, w_4)

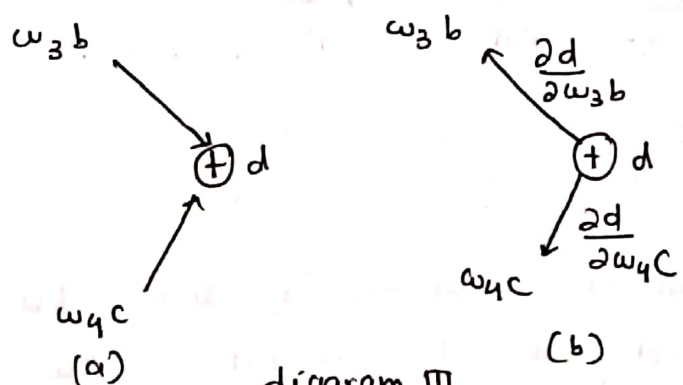
$\downarrow$

d, b and c are created as a result of mathematical operations.

*) computing the gradients:

each node of the graph except leafs, can be considered as a fn. which takes some i/ps and produces an o/p. for example.

$d = f(w_3 b, w_4 c)$



To compute the specific gradient, we mark the incoming gradient coming from the part of the network in front of it.

diagram III

•) This is done for the entire graph as shown in diagram II

•) To compute the gradient then, we follow the steps bellow, example wrt Loss for w_4

i) trace the possible paths from L to w_4.

ii) only one path

iii) multiply all edges along this path.

$$\therefore \frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_4}$$

similarly for $\frac{\partial L}{\partial a} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial b} * \frac{\partial b}{\partial a} + \frac{\partial L}{\partial d} * \frac{\partial d}{\partial c} * \frac{\partial c}{\partial a}$

Note: for different possible, compute along each and add them.

*) autograd in pyTorch :

i) to be included in the graph, the node should have its requires_grad attribute to the tensor to be true. which sometimes might happen implicitly. and sometimes have to be done explicitly.

ii) requires_grad is contagious. means when a tensor is created by operating on other tensors, the requires grad of the resultant tensor would be set to true given at least one of the tensors used for creation has it's requires_grad set to be True.

iii) Also, each tensor has an attribute called grad_fn. which refers to the mathematical operator that created the variable.

↓

If requires_grad = false, grad_fn = None

↓

None also for leaf nodes.

↓

a. grad_fn().

iv) all mathematical operations in pyTorch implemented by the torch.nn.Autograd.function class. two member fns. that are important : forward & backward.

v) forward fn. simply computes the o/p using its inputs.

vi) backward fn. takes the incoming gradient coming from the part of the network in front of it.

↓

gradient to be backpropagated from a function f is basically the gradient that is backpropogated to f from the layers in front of it, multiplied by the local gradient of the o/p of the fn. wrt inputs.

vii) for example. $d = f(w_3 b, w_4 c)$

a) d is our tensor here. grad-fn : <ThAddBackward>. Addition operation.

b) forward fn. receives the inputs $w_3 b$ & $w_4 c$ and adds them. value stored in d.

c) backward fn. simply takes the incoming gradients from further layers as i/p. (stored in grad attribute of d.)

d) local gradients $\frac{\partial d}{\partial w_4 c}$ and $\frac{\partial d}{\partial w_3 b}$.

e) backward fn. multiplies the incoming gradient with locally computed gradients and sends the grads to its i/ps.

•) backward can only be called on a scalar tensor.

•) with torch.no_grad() can be used while performing inference, and thus not storing the additional values.