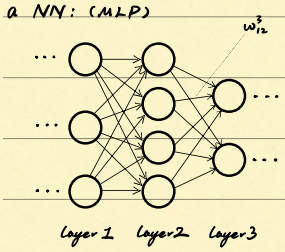
On Backpropagation

Goal: $\frac{\partial C}{\partial w}$ (or $\frac{\partial C}{\partial b}$) w.r.t. any weight w (or bias b) ; where C stands for Cost function

- How quickly the cost changes when we change the w & b
- Overall: BP gives us detailed insights into how changing the w&b changes overall behaviour of the network

notations

a NN: (MLP)

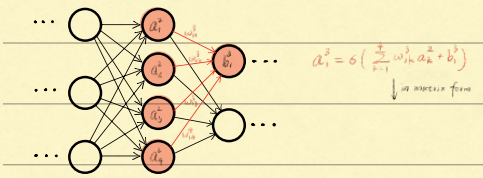


w^l_{jk} : connection from the kth neuron in the $(l-1)$-th layer to the jth neuron in the lth layer

b^l_j : bias of the jth neuron in lth layer

a^l_j : activation of the jth neuron in lth layer

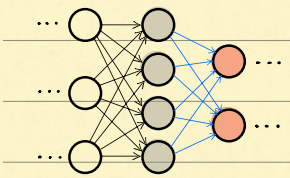
we have : $a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$ σ : activation function (e.g. Sigmoid)



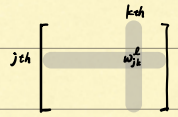
$a^3_1 = \sigma\left(\sum_{k=1}^{4} w^3_{1k} a^2_k + b^3_1\right)$

↓ in matrix form

vectorization :



w^l : weight matrix , weights connecting to the jth layer of neurons



$w^3 = \begin{bmatrix} w^3_{11} & w^3_{12} & w^3_{13} & w^3_{14} \\ w^3_{21} & w^3_{22} & w^3_{23} & w^3_{24} \end{bmatrix}$

$b^2 = \begin{bmatrix} b^2_1 \\ b^2_2 \\ b^2_3 \\ b^2_4 \end{bmatrix}$ $a^3 = \begin{bmatrix} a^3_1 \\ a^3_2 \end{bmatrix}$

$\sigma\left(\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}\right) = \begin{bmatrix} \sigma(v_1) \\ \vdots \\ \sigma(v_m) \end{bmatrix}$

b^l : bias vector , components of the bias vector are values of b^l_j

a^l : activation vector , components of the activation vector are values of a^l_j

σ^l : activation (element-wise) : apply a function σ to every element in vector v

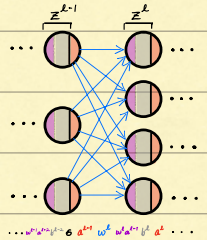
we have : $a^l = \sigma(w^l a^{l-1} + b^l) = \sigma(z^l)$

matrix mul, bias term, activation

$z^l := w^l a^{l-1} + b^l$ (intermediate quantity) $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$,

z^l is "weighted input" to neurons in layer l

So far so good :



two assumptions we need about the cost function

example cost function :

$$C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2$$

cost average over all training samples

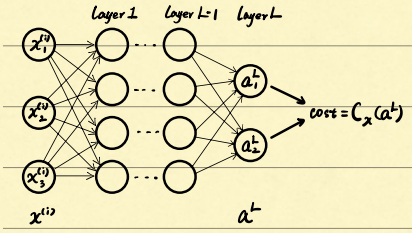
n : total number of training samples $\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{bmatrix}, \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \right)$

x : individual traing sample : i-th sample $x^{(i)}$ $\begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$

$y(x)$: sample x's corresponding desired output (ground-truth / label ...) . $\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$

L : number of layers in the network

$a^L = a^L(x)$: output vector from the network when x is input



1st assumption : cost function C can be written as an average $C = \frac{1}{n} \sum_x C_x$

over functions C_x for individual training sample x

$$C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2 = \frac{1}{n} \sum_x \underbrace{\frac{1}{2} \| y(x) - a^L(x) \|^2}_{C_x}$$

BP computes $\frac{\partial C_x}{\partial w}$ ($\frac{\partial C_x}{\partial b}$) for a single sample

then we recover $\frac{\partial C}{\partial w}$ ($\frac{\partial C}{\partial b}$) by averaging over training sample

2nd assumption : the cost can be written as a function of the outputs from the NN.

So far so good : $C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2$ is a fuction of our NN , as x, $y(x)$ are fixed parmeters.

when we change weight & bias (the parameters in NN) , we'll change the output of NN, given x as input

The four fundamental equations behind backpropagation

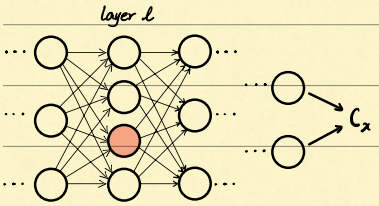
recall: our goal is $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$,

ultimately, we need to compute partial derivatives $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$

to do this, we introduce intermediate quantity δ^l_j : error in the j-th neuron in the l-th layer

BP 1° gives us a procedure to compute the error δ^l_j,

2° and then will relate δ^l_j to $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$



Consider j-th neuron in layer l : ,

given a sample point x, we have weighted input z^l_j, and output $\sigma(z^l_j)$

now add a small change Δz^l_j to z^l_j, the output will be $\sigma(z^l_j+\Delta z^l_j)$

we can view a^l_j as well, z^l_j for notational convenience

this change propagates through later layers in the network,

and finally causing the overall cost to change by an amount $\frac{\partial C}{\partial z^l_j}\Delta z^l_j$

(approx. in this case, Δz^l_j is a small value)

from $\frac{\partial C}{\partial z^l_j}\Delta z^l_j$, we can see that $\frac{\partial C}{\partial z^l_j}$ affects the magnitude of Δz^l_j

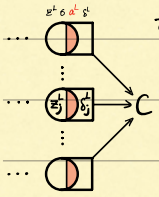
required to change the overall cost by the same amount

by this heuristic, we define $\delta^l_j := \frac{\partial C}{\partial z^l_j}$ as "a measure of the error in the neuron"

different meaning from "error" in a test (3.5% error) for me, it's more of a "rate" thing.

δ^l : the vector of errors associated with layer l.

(BP1) An equation for error in the output layer, δ^L



the components of δ^L are given by

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

measures how fast the cost is changing as a function of the j-th output activation

measures how fast the activation function σ is changing at z^L_j

remember $C = C(a^L_j(z^L_j))$

Pf: Applying the chain rule :

$$\delta^L_j = \frac{\partial C}{\partial z^L_j} = \sum_k \frac{\partial C}{\partial a^L_k}\frac{\partial a^L_k}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\cdot\frac{\partial a^L_j}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- z^L_j can be obtained by one forward-pass on NN
- $\sigma'(z^L_j)$ is easy to compute
- $\frac{\partial C}{\partial a^L_j}$ is easy to compute as the form of cost function C is provided (e.g. $C=\frac{1}{2}\|y(x)-a^L(x)\|^2$)

write BP1 in matrix-based form:

$\delta^L = \nabla_a C \odot \sigma'(z^L)$, where $\odot$ stands for Hadamard product (elementwise product)

or in mat.mul form :

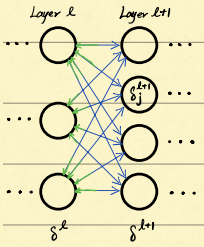
$$\delta^L = \begin{bmatrix} \sigma'(z^L_1) & & 0 \\ & \ddots & \\ 0 & & \sigma'(z^L_j) \end{bmatrix}\cdot\nabla_a C$$

$$= \Sigma'(z^L)\cdot\nabla_a C$$

- $\nabla_a C$: gradient of C w.r.t. the output activations a^L $\quad [\cdots, \frac{\partial C}{\partial a^L_j}, \cdots]$

e.g. for $C=\frac{1}{2}\|y(x)-a^L(x)\|^2$, we have $\delta^L = (a^L-y)\odot\sigma'(z^L)$

(BP2) An equation for the error δ^l in terms of the errors in the next layer, δ^{l+1}



In particular:

$$\delta^l = \left[(w^{l+1})^T \delta^{l+1}\right] \odot \sigma'(z^l)$$

"moving the error backward through the network" gives us some kind of measure of error at the output of the l^{th} layer

or in mat mul form: $\delta^l = \Sigma'(z^l)(w^{l+1})^T \delta^{l+1}$

Pf:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

$$= \sum_k \frac{\partial}{\partial z_j^l}\left[\sum_i w_{ki}^{l+1} \sigma(z_i^l) + b_k^{l+1}\right] \delta_k^{l+1}$$

$$= \sum_k w_{kj}^{l+1} \sigma'(z_j^l) \delta_k^{l+1}$$

$$\delta^l = \left[(w^{l+1})^T \delta^{l+1}\right] \odot \sigma'(z^l)$$

By combining BP1 and BP2 we can compute the error δ^l for any layer in the network

1° compute δ^L using BP1

2° compute $\delta^{L-1}, \delta^{L-2}, \dots$ using BP2

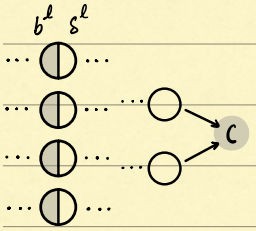
$$\delta^L = \Sigma'(z^L) \nabla_a C$$

$$\delta^{L-1} = \Sigma'(z^{L-1})(w^L)^T \delta^{L+1}$$

$$= \Sigma'(z^{L-1})(w^L)^T \Sigma'(z^L) \nabla_a C$$

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \cdots \Sigma'(z^{L-1})(w^L)^T \Sigma'(z^L) \nabla_a C$$

(BP3) An equation for the rate of change of the cost w.r.t. any bias in the network



In particular:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

δ_j^l is exactly equal to the rate of change $\frac{\partial C}{\partial b_j^l}$!

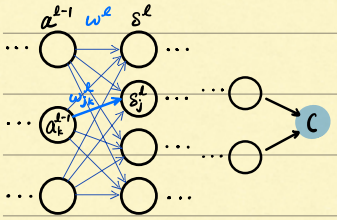
Pf:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l}$$

$$= \delta_j^l \cdot \frac{\partial}{\partial b_j^l}\left[\sum_k w_{jk}^l \sigma(z_i^{l-1}) + b_k^l\right]$$

$$= \delta_j^l$$

(BP4) An equation for the rate of change of the cost w.r.t. any weights in the network



In particular:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Pf:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l}$$

$$= \delta_j^l \cdot \frac{\partial}{\partial w_{jk}^l}\left[\sum_k w_{jk}^l a_k^{l-1} + b_k^l\right]$$

$$= \delta_j^l a_k^{l-1}$$

Summary: equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP 1)}$$

$$\delta^l = \left[(w^{l+1})^T \delta^{l+1}\right] \odot \sigma'(z^l) \qquad \text{(BP 2)}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j \qquad \text{(BP 3)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \qquad \text{(BP 4)}$$

- **Some insights on learning with BP**

in BP1: term $\sigma'(z_j^L) \to 0$ when $z_j^L \to \infty$ (in this case, σ is sigmoid function): a weight in the final layer will learn slowly if the neuron is either low activation (~ 0) or high activation (~ 1), it's common to say the output neuron has saturated, as a result the weight has stopped learning (or learning slowly)

in BP2: similar for earlier layers: δ_j^l is likely to get small if the neuron is near saturation

in BP4: when a^{l-1} is small, $\frac{\partial C}{\partial w^l}$ also tends to be small

affect ↓

in context of gradient decent, the weights output from low-activation neurons learn slowly

- **we can turn the observations around:**

e.g. activation function σ s.t. σ' is always positive & never gets close to zero

this would prevent the slow-down of learning that occurs when ordinary sigmoid neurons saturate

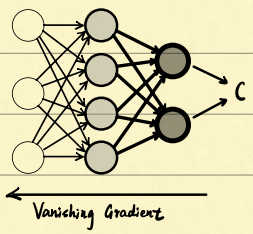
- **vanishing/exploding gradient:**

Consider error vector δ^l, we have $\delta^l = \Sigma'(z^l)(w^{l+1})^T \cdots \Sigma'(z^{l-1})(w^L)^T \Sigma'(z^L) \nabla_a C$

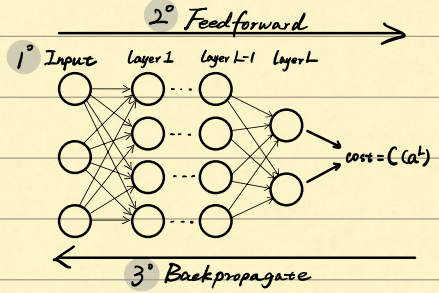
notably, there are multiple products of these derivatives of the activation function $(\Sigma'(z^l))$

this observation and the insights above will lead to the problem of the vanishing/exploding gradient

which is a fundamental issue arising from NN's architecture and the choice of activation functions



The Backpropagation algorithm



1° Input x: set the corresponding activation a^1 for the input layer

2° Feedforward: For each $l = 2, 3, \dots, L$, compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$

3° Output error δ^L: Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$

4° Backpropagate the error: For each $l = L-1, L-2, \dots, 2$, compute $\delta^l = [(w^{l+1})^T \delta^{l+1}] \odot \sigma'(z^l)$

5° Output: The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

Combining BP with mini-batch SGD:

given a mini-batch of m training examples:

1° Input a set of training examples

2° for each training example x:

- set the corresponding activation $a^{x,1}$ for the input layer
- Feedforward: For each $l = 2, 3, \dots, L$, compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$
- Output error $\delta^{x,L}$: Compute the vector $\delta^{x,L} = \nabla_a C \odot \sigma'(z^{x,L})$
- Backpropagate the error: For each $l = L-1, L-2, \dots, 2$, compute $\delta^{x,l} = [(w^{l+1})^T \delta^{x,l+1}] \odot \sigma'(z^{x,l})$

3° Gradient descent: For each $l = L, L-1, \dots, 2$,

update the weights: $w^l \to w^l - \eta \cdot \frac{1}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$

biases: $b^l \to b^l - \eta \cdot \frac{1}{m} \sum_x \delta^{x,l}$

matrix-based approach

represent a mini-batch input by a matrix $X = [x^{(1)}, \dots, x^{(m)}] = \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(m)} \\ \vdots & & \vdots \\ x_n^{(1)} & & x_n^{(m)} \end{bmatrix}$

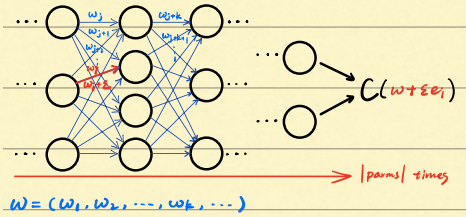
$$a' = \left[w' \cdot \sigma(X) + b' \right]$$

introduce a new axis "$\leftrightarrow$" $\begin{bmatrix} \cdots & a^{x,1}_j & \cdots \end{bmatrix}_{out \times m} = \begin{bmatrix} w^1_{ij}, \dots, w^1_{nj} \end{bmatrix}_{out \times n} \begin{bmatrix} \sigma(x_1^{(1)}) & \cdots & \sigma(x_1^{(m)}) \\ \vdots & & \vdots \\ \sigma(x_n^{(1)}) & \cdots & \sigma(x_n^{(m)}) \end{bmatrix}_{n \times m} + \begin{bmatrix} b^1_j & \cdots & b^m_j \end{bmatrix}_{out \times m}$

$$A^l_{out \times m} = \sigma\left(w^l_{out \times in} A^{l-1}_{in \times m} + B^l_{out \times m} \right)$$

In what sense is backpropagation a fast algorithm

another approach to computing the gradient:



still, our goal is computing $\frac{\partial C}{\partial w_j}$ for every j.

by definition, we have $\frac{\partial C}{\partial w_j} = \lim_{\varepsilon \to 0^+} \frac{C(w+\varepsilon e_j) - C(w)}{\varepsilon}$, where e_j is the unit vector in j-th direction.

now we can estimate $\frac{\partial C}{\partial w_j}$ by $\frac{C(w_j) - C(w_j')}{w_j - w_j'}$, where w_j, w_j' are 2 slightly differer values of w_j

and it's same for $\frac{\partial C}{\partial b}$.

a promising method, but the implementation turns out to be extremely slow.

e.g. a NN with million weights. then for each w_j, we need to compute $C(w+\varepsilon e_j)$

thus we needs to compute the cost function a million different times, requiring a million forward passes through the network (per training example)!

In contrast, backpropagation enables us to compute all partial derivatives $\frac{\partial C}{\partial w_j}$

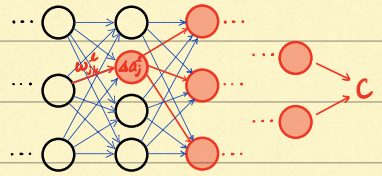
using just one forward pass through the network,

followed by one backward pass through the network.

Backpropagation : the big picture

more intuition — how changing the w & b changes overall behaviour of the network?

when we add a small change to w^l_{jk}, $w^l_{jk} \to w^l_{jk} + \Delta w^l_{jk}$, this will cause changes in following layers



now focus on the change ΔC in the cost, $\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk}$

this suggest that a possible approach to computing $\frac{\partial C}{\partial w^l_{jk}}$:

track how a small change in w^l_{jk} propagates to cause a small change in C.

Δw^l_{jk} causes a small change Δa^l_j in the activation of the j-th neuron in the l-th layer :

$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

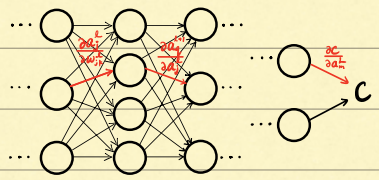
Δa^l_j will cause changes in all the activations in the next layer, consider the q-th neuron :

$$\Delta a^{l+1}_q \approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \Delta a^l_j$$

$$\approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

now consider a single "path" : $a^l_j, a^{l+1}_q, \cdots, a^{L-1}_n, a^L_m$, we have

$$\Delta C_0 \approx \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$



for total change in C :

$$\Delta C \approx \sum_{m,n,\cdots,q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

$$\Rightarrow \frac{\partial C}{\partial w^l_{jk}} = \sum_{m,n,\cdots,q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}}$$

$$= \sum_{m,n,\cdots,q} \frac{\partial C}{\partial a^L_m} \sigma'(z^L_m) w^L_{mn} \cdot \sigma'(z^{L-1}_n) w^{L-1}_{np} \cdots \sigma'(z^{l+1}_q) w^{l+1}_{qj} \cdot \sigma'(z^l_j) a^{l-1}_k$$

$$= \sum_m \frac{\partial C}{\partial a^L_m} \sigma'(z^L_m) \sum_n w^L_{mn} \cdot \sigma'(z^{L-1}_n) \sum_p \cdots \sum_q w^{l+2}_{qj} \sigma'(z^{l+1}_q) \cdot w^{l+1}_{qj} \sigma'(z^l_j) \cdot a^{l-1}_k$$

$$= \sum_m \delta^L_m \sum_n w^L_{mn} \cdot \sigma'(z^{L-1}_n) \sum_p \cdots \sum_q w^{l+2}_{qj} \sigma'(z^{l+1}_q) \cdot w^{l+1}_{qj} \sigma'(z^l_j) \cdot a^{l-1}_k$$

$$= \sum_n \sum_m w^L_{mn} \cdot \sigma'(z^{L-1}_n) \delta^L_m \sum_p \cdots \sum_q w^{l+2}_{qj} \sigma'(z^{l+1}_q) \cdot w^{l+1}_{qj} \sigma'(z^l_j) \cdot a^{l-1}_k$$

$$= \sum_n \delta^{L-1}_n \sum_p \cdots \sum_q w^{l+2}_{qj} \sigma'(z^{l+1}_q) \cdot w^{l+1}_{qj} \sigma'(z^l_j) \cdot a^{l-1}_k$$

$$= \sum_q \delta^{l+1}_q \cdot w^{l+1}_{qj} \sigma'(z^l_j) \cdot a^{l-1}_k$$

$$= \delta^{l-1}_k a^{l-1}_k$$

this is exactly BP 2 !

In this context, BP is a "layer-wise" order to compute the summation !