All The BPs

CMPUT 328

Nilanjan Ray

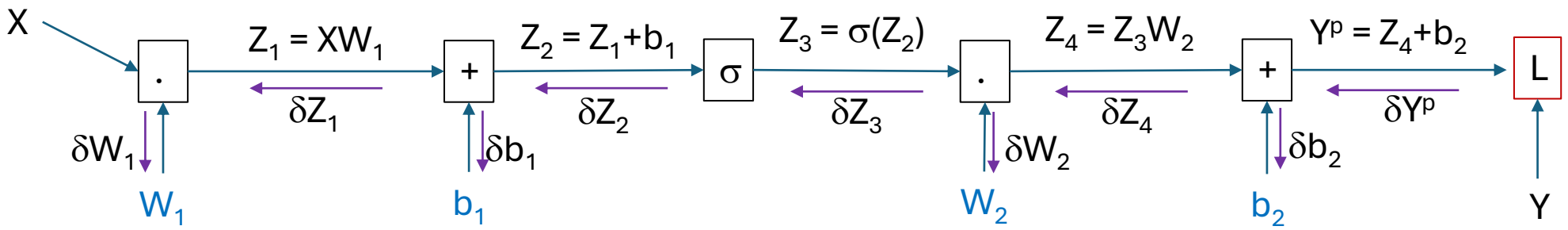
OK, let's apply chain rule to a computational graph where all variables and parameters are scalars

So, our scalar neural net is: $Y^p = \sigma(XW_1 + b_1)W_2 + b_2$

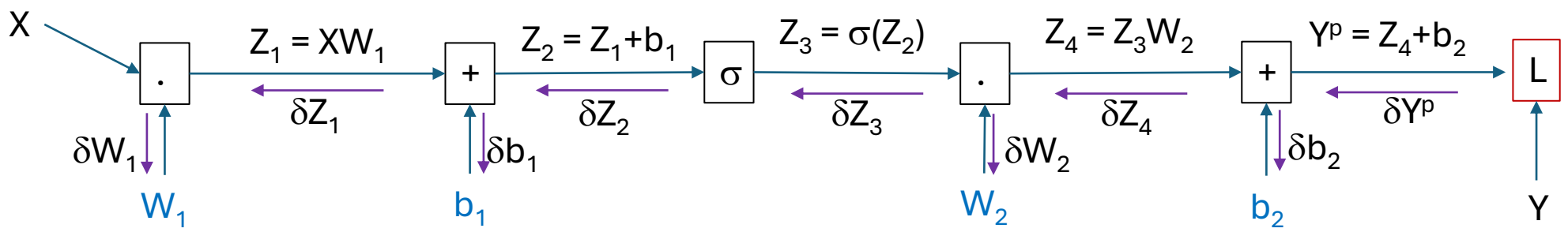
with a square (aka Euclidean) loss function: $L(Y^p, Y) = \frac{1}{2}(Y^p - Y)^2$

As usual, X is the input, Y^p is the output, and W_1, b_1, W_2, b_2 are parameters of the neural net, σ is a non-linear function.

The computational graph for this scalar neural net is (also showing loss gradient symbols):



Chain rule for a scalar neural net…



1 $$\delta Y^p \equiv \frac{\partial L}{\partial Y^p} = \frac{\partial}{\partial Y^p}\left[\frac{1}{2}(Y^p - Y)^2\right] = y^p - y$$

2 $$\delta Z_4 \equiv \frac{\partial L}{\partial Z_4} = \frac{\partial Y^p}{\partial Z_4}\frac{\partial L}{\partial Y^p} = \delta Y^p$$

Because $Y^p = Z_4 + b_2, \frac{\partial Y^p}{\partial Z_4} = 1$

3 $$\delta Z_3 \equiv \frac{\partial L}{\partial Z_3} = \frac{\partial Z_4}{\partial Z_3}\frac{\partial L}{\partial Z_4} = W_2\delta Z_4$$

Because $Z_4 = Z_3W_2, \frac{\partial Z_4}{\partial Z_3} = W_2$

4 $$\delta Z_2 \equiv \frac{\partial L}{\partial Z_2} = \frac{\partial Z_3}{\partial Z_2}\frac{\partial L}{\partial Z_3} = \sigma'(Z_2)\delta Z_3$$

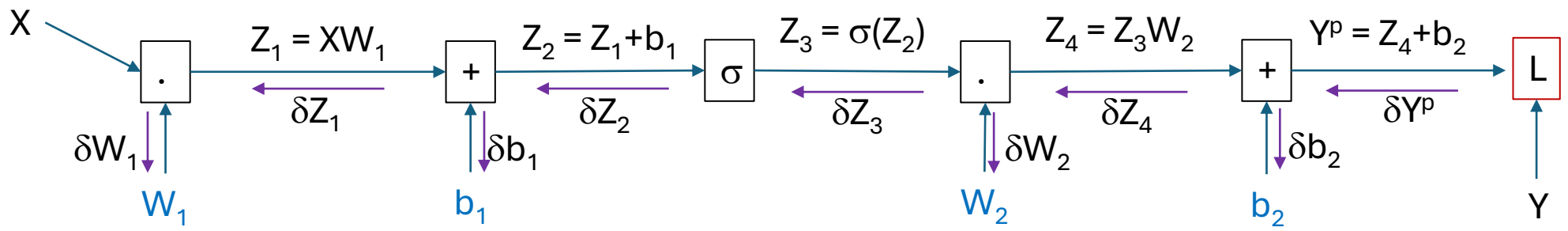
Because $Z_3 = \sigma(Z_2), \frac{\partial Z_3}{\partial Z_2} = \sigma'(Z_2)$

5 $$\delta Z_1 \equiv \frac{\partial L}{\partial Z_1} = \frac{\partial Z_2}{\partial Z_1}\frac{\partial L}{\partial Z_2} = \delta Z_2$$

Because $Z_2 = Z_1 + b_1, \frac{\partial Z_2}{\partial Z_1} = 1$

But we need loss derivatives with respect to parameters…

Loss derivatives w.r.t. parameters



1 $$\delta W_1 \equiv \frac{\partial L}{\partial W_1} = \frac{\partial Z_1}{\partial W_1}\frac{\partial L}{\partial Z_1} = X\delta Z_1$$

Because $Z_1 = XW_1, \frac{\partial Z_1}{\partial W_1} = X$

2 $$\delta b_1 \equiv \frac{\partial L}{\partial b_1} = \frac{\partial Z_2}{\partial b_1}\frac{\partial L}{\partial Z_2} = \delta Z_2$$

Because $Z_2 = Z_1 + b_1, \frac{\partial Z_2}{\partial b_1} = 1$

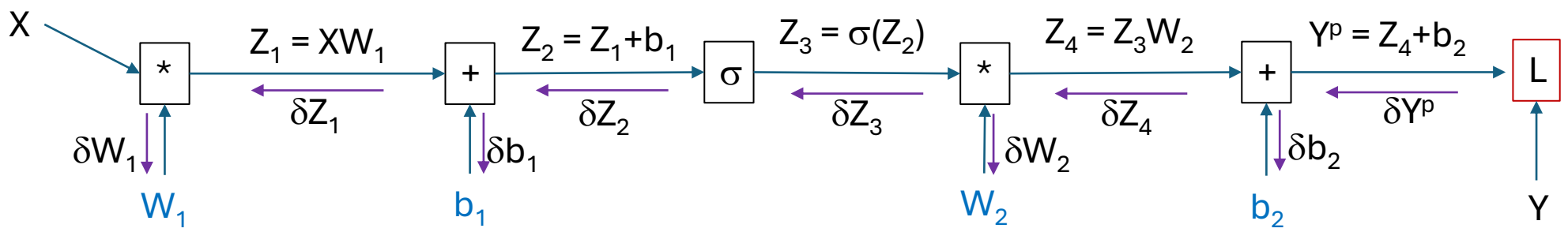
3 $$\delta W_2 \equiv \frac{\partial L}{\partial W_2} = \frac{\partial Z_4}{\partial W_2}\frac{\partial L}{\partial Z_4} = Z_3\delta Z_4$$

Because $Z_4 = Z_3W_2, \frac{\partial Z_4}{\partial W_2} = Z_3$

4 $$\delta b_2 \equiv \frac{\partial L}{\partial b_2} = \frac{\partial Y^p}{\partial b_2}\frac{\partial L}{\partial Y^p} = \delta Y^p$$

Because $Y^p = Z_4 + b_2, \frac{\partial Y^p}{\partial b_2} = 1$

Chain rule for a (general) neural net



1 $$\delta Y^p \equiv \frac{\partial L}{\partial Y^p} = \frac{\partial}{\partial Y^p}\left[\frac{1}{2}||Y^p - Y||^2\right] = y^p - y$$

2 $$\delta Z_4 \equiv \frac{\partial L}{\partial Z_4} = \frac{\partial Y^p}{\partial Z_4}\frac{\partial L}{\partial Y^p} = \delta Y^p$$

Because $Y^p = Z_4 + b_2, \frac{\partial Y^p}{\partial Z_4} = 1$

3 $$\delta Z_3 \equiv \frac{\partial L}{\partial Z_3} = \frac{\partial Z_4}{\partial Z_3}\frac{\partial L}{\partial Z_4} = \delta Z_4 W_2^T$$

Because $Z_4 = Z_3W_2, \frac{\partial Z_4}{\partial Z_3} = W_2^T$

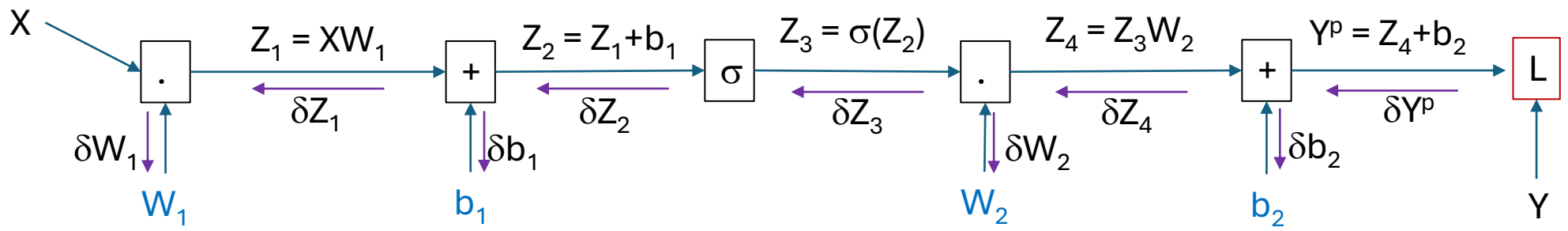
4 $$\delta Z_2 \equiv \frac{\partial L}{\partial Z_2} = \frac{\partial Z_3}{\partial Z_2}\frac{\partial L}{\partial Z_3} = \sigma'(Z_2).\delta Z_3$$

Because $Z_3 = \sigma(Z_2), \frac{\partial Z_3}{\partial Z_2} = \sigma'(Z_2)$

5 $$\delta Z_1 \equiv \frac{\partial L}{\partial Z_1} = \frac{\partial Z_2}{\partial Z_1}\frac{\partial L}{\partial Z_2} = \delta Z_2$$

Because $Z_2 = Z_1 + b_1, \frac{\partial Z_2}{\partial Z_1} = 1$

Loss derivatives w.r.t. matrix or vector parameters



1 $$\delta W_1 \equiv \frac{\partial L}{\partial W_1} = \frac{\partial Z_1}{\partial W_1}\frac{\partial L}{\partial Z_1} = X^T \delta Z_1$$

Because $Z_1 = XW_1, \frac{\partial Z_1}{\partial W_1} = X^T$

2 $$\delta b_1 \equiv \frac{\partial L}{\partial b_1} = \frac{\partial Z_2}{\partial b_1}\frac{\partial L}{\partial Z_2} = \sum_k (\delta Z_2)_{k,:}$$

Because $Z_2 = Z_1 + b_1, \frac{\partial Z_2}{\partial b_1} = [1, \dots, 1]$

3 $$\delta W_2 \equiv \frac{\partial L}{\partial W_2} = \frac{\partial Z_4}{\partial W_2}\frac{\partial L}{\partial Z_4} = Z_3^T \delta Z_4$$

Because $Z_4 = Z_3W_2, \frac{\partial Z_4}{\partial W_2} = Z_3^T$

4 $$\delta b_2 \equiv \frac{\partial L}{\partial b_2} = \frac{\partial Y^p}{\partial b_2}\frac{\partial L}{\partial Y^p} = \sum_k (\delta Y^p)_{k,:}$$

Because $Y^p = Z_4 + b_2, \frac{\partial Y^p}{\partial b_2} = [1, \dots, 1]$

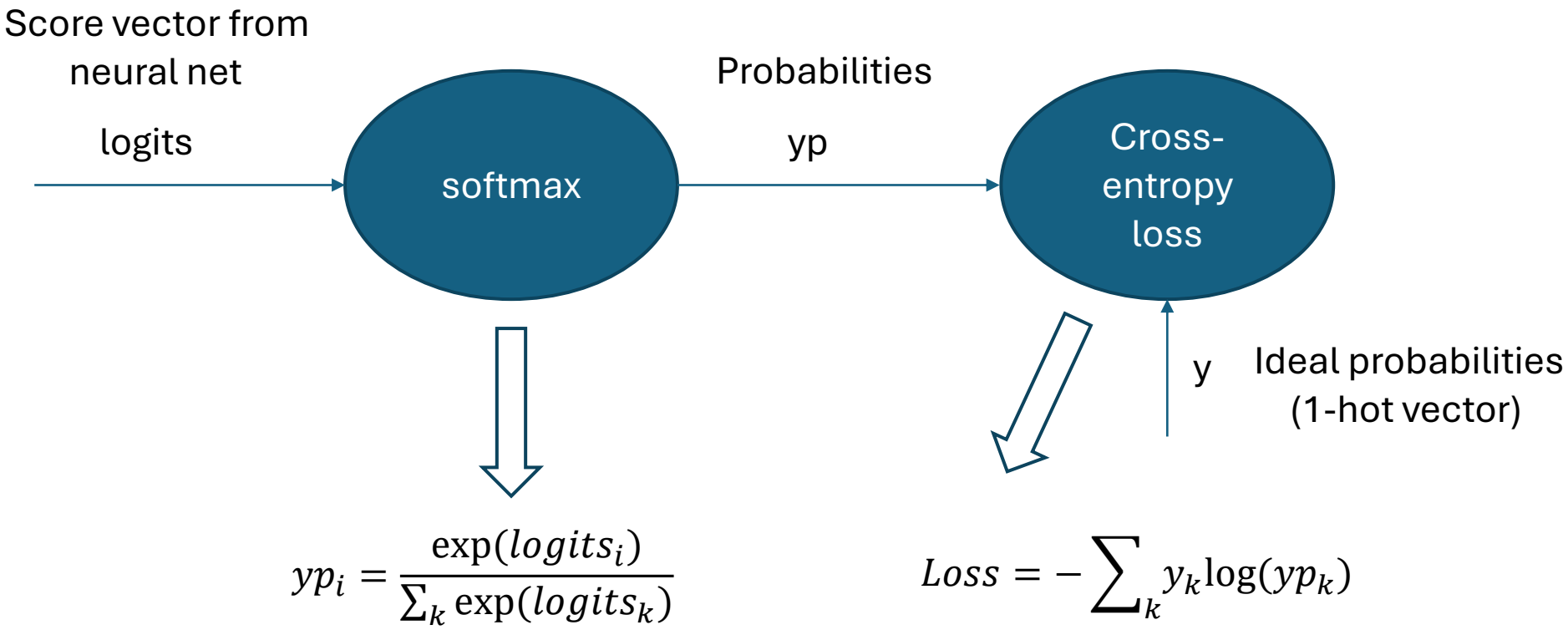
Backprop derivation for loss function

Euclidean loss function: $Loss(Y^p, Y) = \frac{1}{2}\|Y^p - Y\|^2 = \frac{1}{2}\sum_i \left(Y_i^p - Y_i\right)^2$

i^{th} component of δY^p vector: $\delta Y_i^p = \frac{\partial}{\partial Y_i^p} Loss(Y^p, Y) = \frac{\partial}{\partial Y_i^p}\frac{1}{2}\sum_k \left(Y_k^p - Y_k\right)^2 = Y_i^p - Y_i$

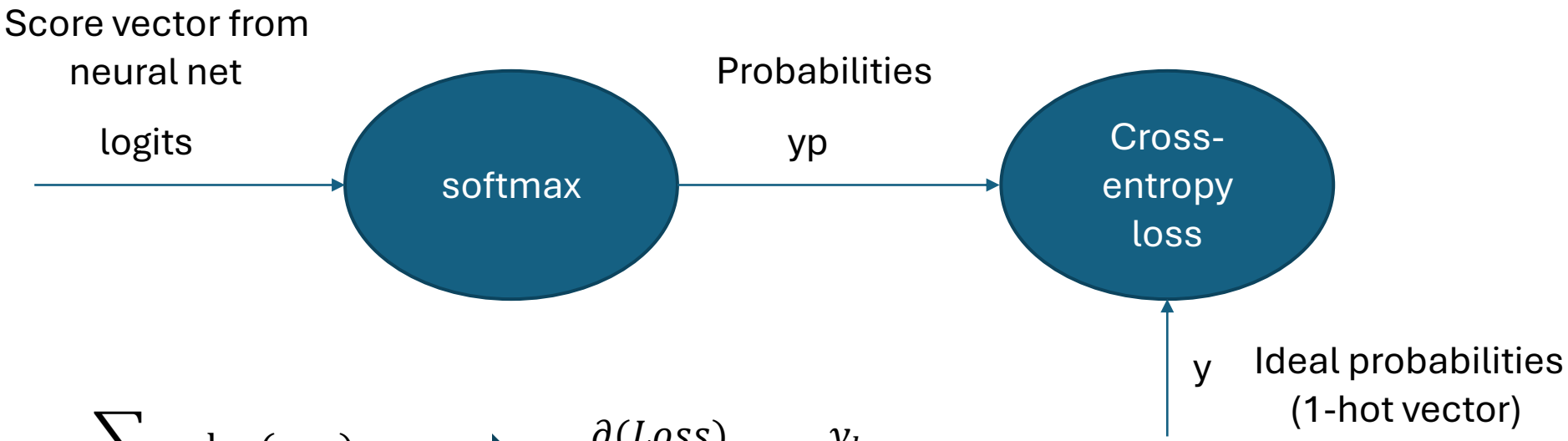
Using vector notation: $\delta Y^p = Y^p - Y$

Softmax and cross-entropy loss



To backpropagate error, we need to compute: $\delta(logits)_i \equiv \frac{\partial(Loss)}{\partial(logits)_i}$

Softmax and cross-entropy loss: backprop



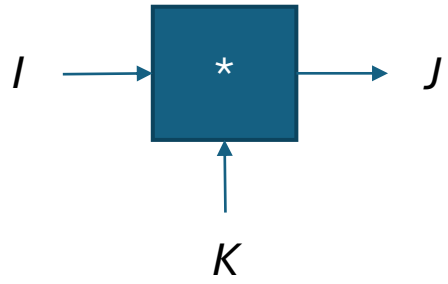
$$Loss = -\sum_k y_k \log(yp_k) \Rightarrow \frac{\partial(Loss)}{\partial(yp)_k} = -\frac{y_k}{yp_k}$$

$$yp_i = \frac{\exp(logits_i)}{\sum_k \exp(logits_k)} \Rightarrow \frac{\partial(yp)_k}{\partial(logits)_i} = \begin{cases} yp_i(1 - yp_i), & \text{if } i = k, \\ -yp_i yp_k, & \text{otherwise.} \end{cases}$$

Using the above two results in the chain rule, $\delta(logits)_i \equiv \frac{\partial(Loss)}{\partial(logits)_i} = \sum_k \frac{\partial(yp)_k}{\partial(logits)_i} \frac{\partial(Loss)}{\partial(yp)_k} = yp_i - y_i$

What if, instead of cross-entropy, we used L2 loss along with softmax?

Backpropagation (BP) for a conv layer



$$J = I * K$$

Input to convolution layer: *I*, a *H*-by-*W* matrix

Parameter of the layer: *K*, a *h*-by-*w* matrix

Output of the layer: *J*, a (*H*-*h*+1)-by-(*W*-*w*+1) matrix

Assume: *H* >= *h* and *W* >= *w*

Given the gradient of loss function δJ with respect to *J*, BP tries to find answers to the following:

(1) What is the gradient of the loss function with respect to *K*? Denote this gradient by δK.

(2) What is the gradient of the loss function with respect to *I*? Denote this gradient by δI.

Why do we need δK? Because, we want to adjust the parameter *K* by gradient descent: *K* = *K* – (learning rate)δK

Why do we need δI? Because, we want to apply BP to the layer that precedes this conv layer.

Derivation of δK

$$J(i,j) = \sum_{l=1}^{h} \sum_{m=1}^{w} I(i+l-1, j+m-1) K(l,m) \quad \Rightarrow \quad \frac{\partial J(i,j)}{\partial K(p,q)} = I(i+p-1, j+q-1)$$

Using chain rule of derivative:

$$\delta K(p,q) = \sum_{i=1}^{H-h+1} \sum_{j=1}^{W-w+1} \frac{\partial J(i,j)}{\partial K(p,q)} \delta J(i,j) = \sum_{i=1}^{H-h+1} \sum_{j=1}^{W-w+1} I(i+p-1, j+q-1) \delta J(i,j)$$

Thus, $\boxed{\delta K = I * \delta J}$

Derivation of δI

$$J(i,j) = \sum_{l=1}^{h} \sum_{m=1}^{w} I(i+l-1, j+m-1) K(l,m)$$

$$\frac{\partial J(i,j)}{\partial I(p,q)} = \begin{cases} K(p-i+1, q-j+1), & \text{if } 0 \le p-i \le h-1 \text{ and } 0 \le q-j \le w-1, \\ 0, & \text{otherwise.} \end{cases}$$

Using chain rule of derivative:

$$\delta I(p,q) = \sum_{i=1}^{H-h+1} \sum_{j=1}^{W-w+1} \frac{\partial J(i,j)}{\partial I(p,q)} \delta J(i,j) = \sum_{i=\max(1,p-h+1)}^{\min(p,H-h+1)} \sum_{j=\max(1,q-w+1)}^{\min(q,W-w+1)} K(p-i+1, q-j+1) \delta J(i,j)$$

$$= \sum_{l=\max(1,p+h-H)}^{\min(p,h)} \sum_{\max(1,q+w-W)}^{\min(q,w)} K(l,m) \delta J(p-l+1, q-m+1)$$

Thus, $\delta I = \text{pad}(\delta J) * \text{flip}(K)$

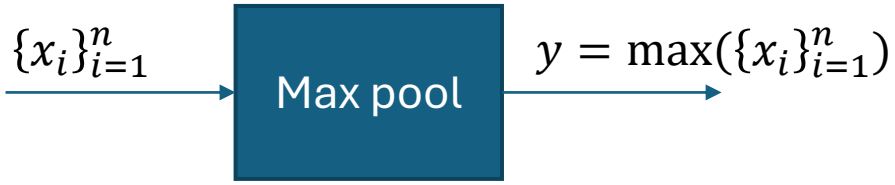
"pad" function adds (h-1) 0 rows at the top and bottom and also adds (w-1) 0 columns at the left and at the right of a matrix.

Size of pad(δJ) is (H+h-1)-by-(W+w-1).

flip(K) is best understood by an example:

$$K = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \qquad \text{flip}(K) = \begin{bmatrix} 6 & 4 & 2 \\ 5 & 3 & 1 \end{bmatrix}$$

BP for a max pooling layer



Note that in case of a tie, only a single index is chosen for the following operation:

$$i = \text{argmax}_k \{x_k\}_{k=1}^n$$

By chain rule: $\delta x_i = \dfrac{\partial y}{\partial x_i}\delta y = \begin{cases} \delta y, & \text{if} \quad i = \text{argmax}_k \{x_k\}_{k=1}^n, \\ 0, & \text{otherwise.} \end{cases}$

Example of a 2-by-2, stride 2 max pooling:

$$x = \begin{bmatrix} 2 & -3 & 6 & 5 \\ 4 & 5 & 6 & -7 \\ 0 & -1 & 3 & -4 \\ -2 & -6 & 6 & 8 \end{bmatrix} \rightarrow \boxed{\text{Max pool}} \rightarrow y = \begin{bmatrix} 5 & 6 \\ 0 & 8 \end{bmatrix}$$

Suppose, $\delta y = \begin{bmatrix} \delta y_1 & \delta y_3 \\ \delta y_2 & \delta y_4 \end{bmatrix}$,

then, $\delta x = \begin{bmatrix} 0 & 0 & \delta y_3 & 0 \\ 0 & \delta y_1 & 0 & 0 \\ \delta y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \delta y_4 \end{bmatrix}$.

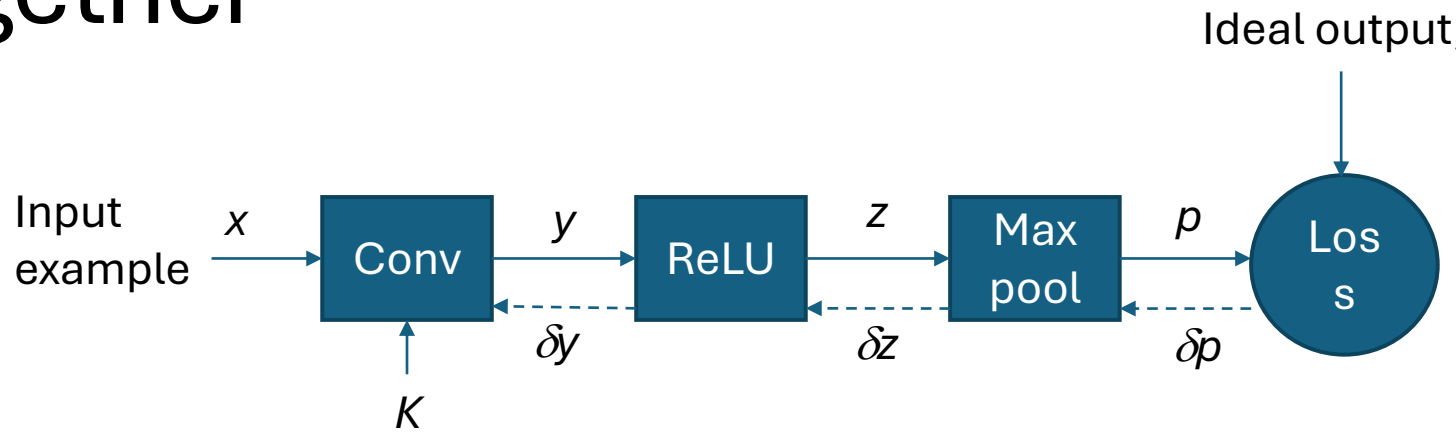
BP for a ReLU layer

By chain rule: $\delta x = \frac{\partial y}{\partial x}\delta y = \begin{cases} \delta y, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases}$

Example: $x = \begin{bmatrix} -5 & 4 \\ 0 & 2 \end{bmatrix}$ → ReLU → $y = \begin{bmatrix} 0 & 4 \\ 0 & 2 \end{bmatrix}$

Suppose, $\delta y = \begin{bmatrix} \delta y_1 & \delta y_3 \\ \delta y_2 & \delta y_4 \end{bmatrix}$, then $\delta x = \begin{bmatrix} 0 & \delta y_3 \\ 0 & \delta y_4 \end{bmatrix}$.

Putting it all together



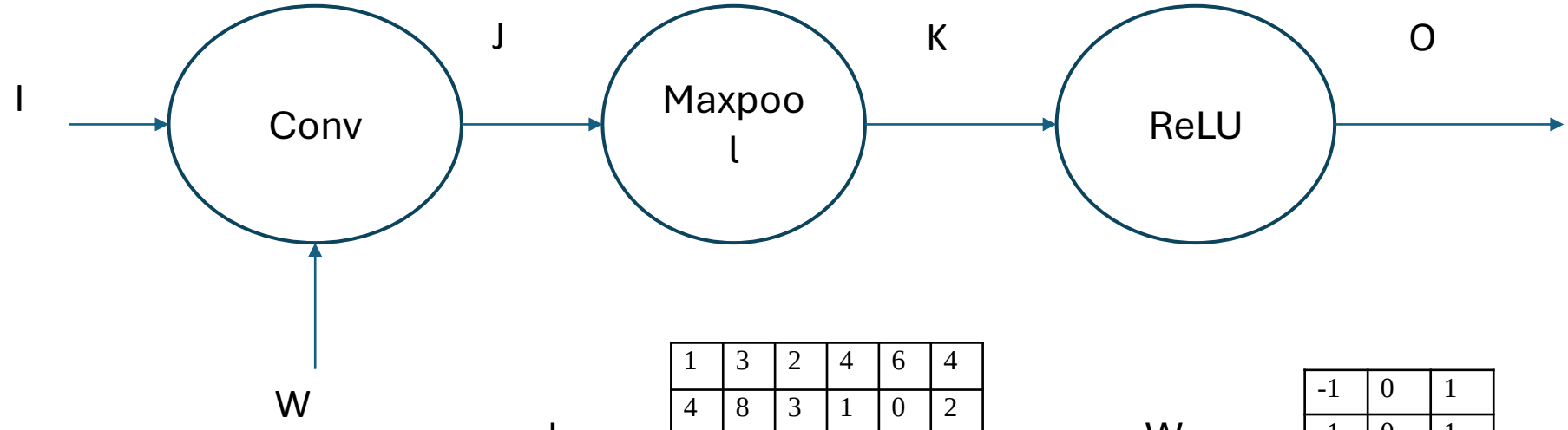
An example network with 3 layers and a loss function

Convnet training algorithm:

Initialize parameter K.
Iterate:
Step 1: (Forward pass)
Step 1a: Randomly choose a training example x and its corresponding ideal output q.
Step 1b: Pass x through "Conv" to get y; pass y though ReLU to get z; pass z through Max pool to get p.
Step 2: Compute "Loss" function for diagnostic purposes. /* Loss function measures deviation of p from q. */
Step 3: (Backward pass aka backpropagation)
Step 3a: Compute gradient of Loss function with respect to p. Denote this gradient by δp.
Step 3b: Compute δz given δp. /* Look at "BP for Max pooling." */
Step 3c: Compute δy given δz. /* Look at "BP for ReLU." */
Step 4c: Compute δK given δy. /* Look at "BP for Conv." */
Step 4: (Update parameter K by gradient descent) $K = K$ – (learning rate) δK.

Note: We don't have to compute δx, because there is no layer preceding conv layer in the example above.

Example w/PyTorch



I =

1	3	2	4	6	4
4	8	3	1	0	2
2	1	4	3	9	1
4	7	2	3	9	2

W =

-1	0	1
-1	0	1
-1	0	1

Compute J, K and O. You are not doing any zero padding while doing the convolution. Assume stride size 1 for the convolution. Assume a 2-by-2 max pooling with stride 2. Write J, K and O below.

Now Assume that ideal output IO = [7, 10]. Also assume the loss as 0.5 times the square of Euclidean distance between IO and O. Compute back-propagation for O, K, J and W. Verify using pytorch autograd!

Practice_Problem.ipyn
b

Example 2

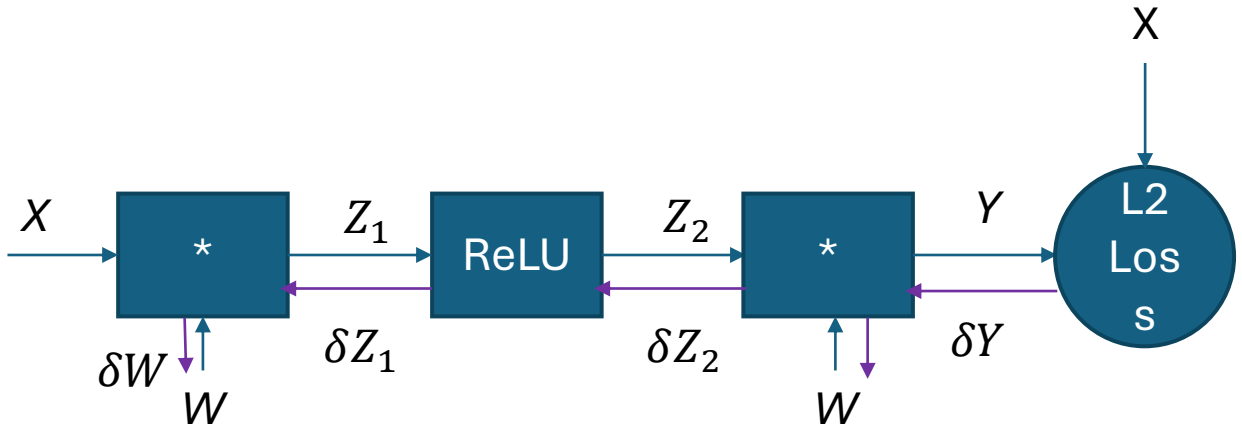
Consider a fully connected neural net:

$$Y = \mathrm{ReLU}(X*W)*W$$

The loss is L2 between Y and X. Compute the gradient of the loss with respect to W. Here ‘*’ refers to matrix multiplication.

Note that the same parameter matrix W appears in two computational nodes – called shared parameters/weights.

Example 2 *solution*



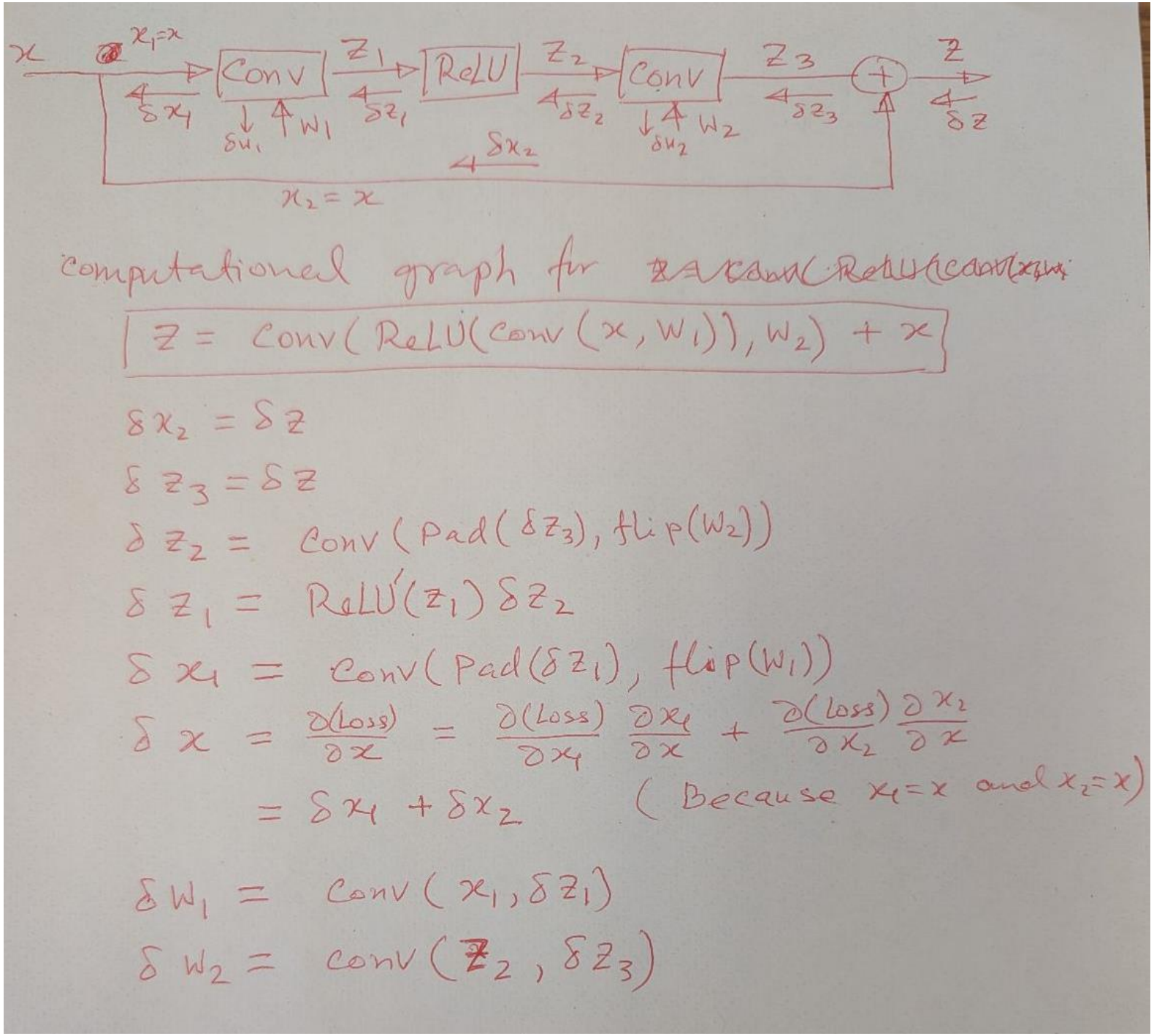
$$\delta Y = Y - X$$

$$\delta Z_2 = \delta Y * W^T$$

$$\delta Z_1 = ReLU'(Z1) * \delta Z_2$$

$$\delta W = {Z_2}^T * \delta Y + X^T * \delta Z_1$$

BP in residual connection



computational graph for $z = \text{Conv}(\text{ReLU}(\text{Conv}(x, W_1)), W_2) + x$

$$z = \text{Conv}(\text{ReLU}(\text{Conv}(x, W_1)), W_2) + x$$

$$\delta x_2 = \delta z$$

$$\delta z_3 = \delta z$$

$$\delta z_2 = \text{Conv}(\text{Pad}(\delta z_3), \text{flip}(W_2))$$

$$\delta z_1 = \text{ReLU}'(z_1)\, \delta z_2$$

$$\delta x_1 = \text{Conv}(\text{Pad}(\delta z_1), \text{flip}(W_1))$$

$$\delta x = \frac{\partial(\text{Loss})}{\partial x} = \frac{\partial(\text{Loss})}{\partial x_1}\frac{\partial x_1}{\partial x} + \frac{\partial(\text{Loss})}{\partial x_2}\frac{\partial x_2}{\partial x}$$

$$= \delta x_1 + \delta x_2 \quad (\text{Because } x_1 = x \text{ and } x_2 = x)$$

$$\delta W_1 = \text{Conv}(x_1, \delta z_1)$$

$$\delta W_2 = \text{Conv}(z_2, \delta z_3)$$

BP in transformer