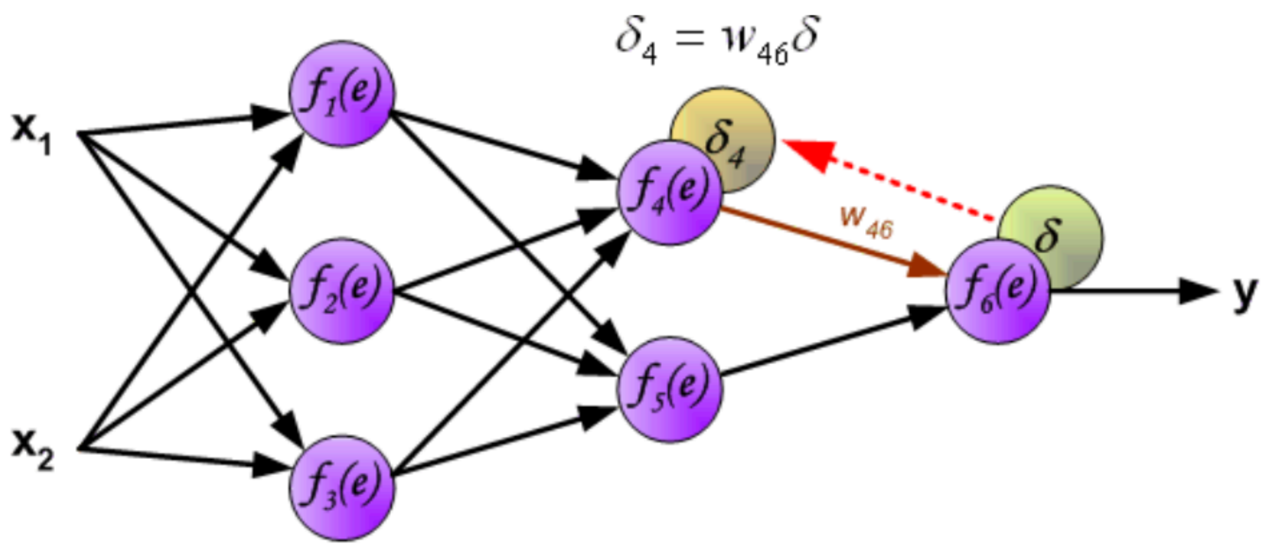
Computer Vision : Insight of object detection algorithms :

Lesson4 Basic Neutral Network



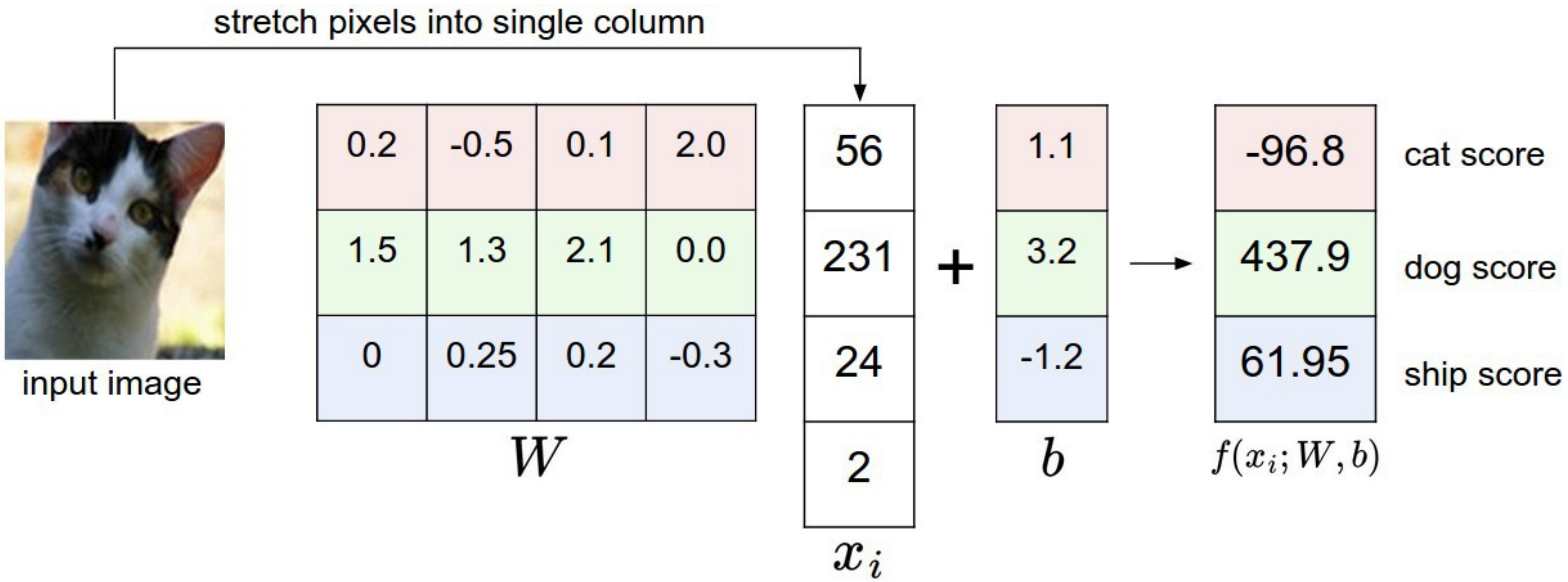
By Daniel
Research Scientist
danielshaving@gmail.com

Table of contents

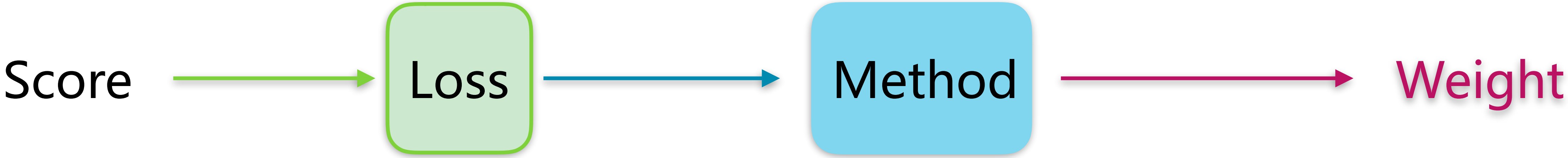
An NN model

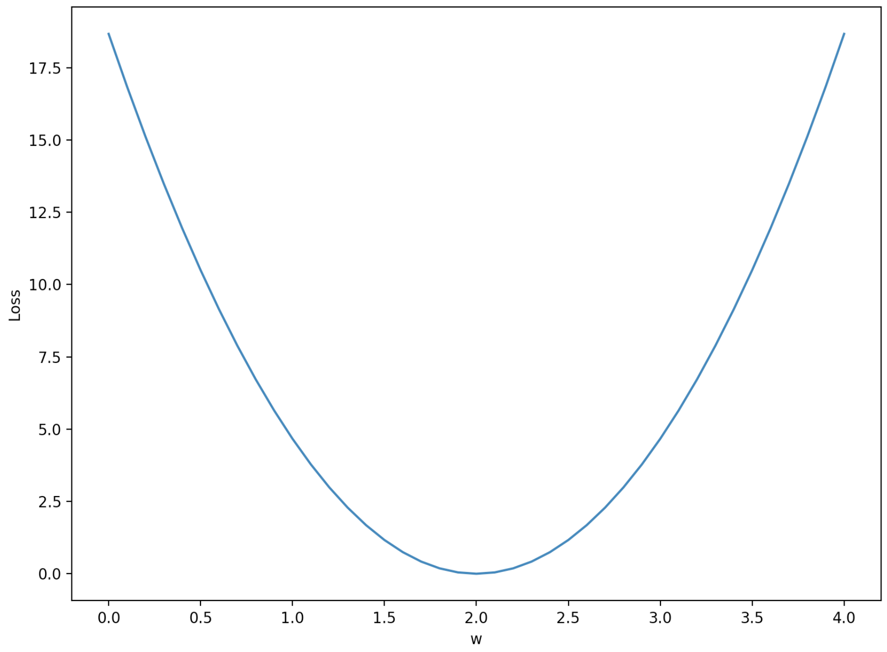
Suppose we have a weights matrix $W \in \mathrm{R}^{K\times D}$, a bias vector $b \in \mathrm{R}^{K\times 1}$ where K is number of classes and D is feature dimensionality, then given an input $x_i \in \mathrm{R}^{D\times 1}$, the score funtion $f : \mathrm{R}^{D} \rightarrow \mathrm{R}^{K}$ is:

$$f(x_i, W, b) = W x_i + b$$



Using score to correct weights (supervising)



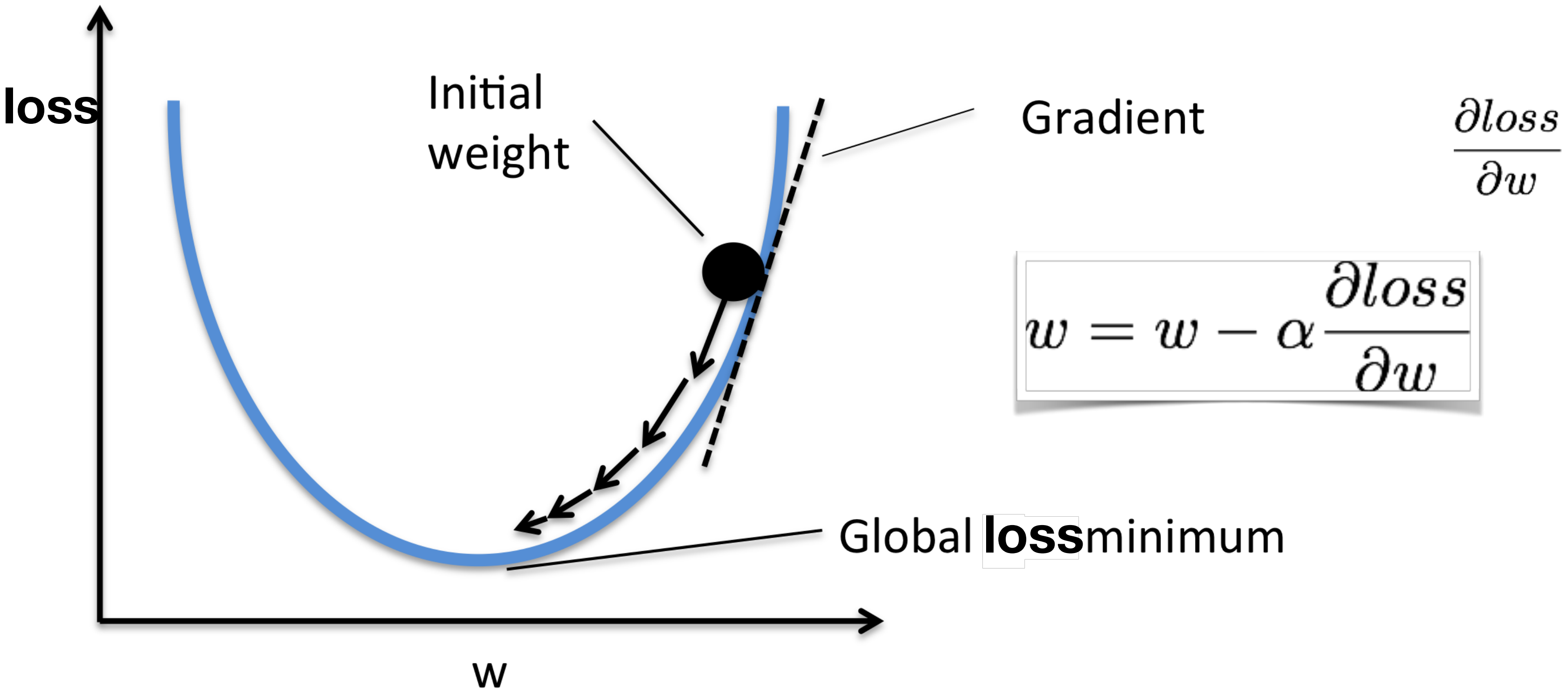


$$loss(w) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y_n} - y_n)^2$$

$$\arg\min_{w} loss(w)$$

Example: UNet $(x \times w - y)^2$

Score → Loss → Method → Weight



Derivative $(x \times w - y)^2$

$$loss = (\hat{y} - y)^2 = (x * w - y)^2$$

$$w = w - \alpha \frac{\partial loss}{\partial w}$$

$$\frac{\partial loss}{\partial w} = ?$$

$$\boxed{w = w - \alpha \frac{\partial loss}{\partial w}}$$

$$= w - \alpha * 2x(xw - y)$$

YOUR INPUT:
$f(w) =$

$$(xw - y)^2$$

Simplify **Roots/zeros**

FIRST DERIVATIVE:
$\frac{d}{dw}[f(w)] = f'(w) =$

The steps of calculation are displayed.
Move the mouse over a derivative $\frac{d}{dw}[\ldots]$ or tap it in order to show its calculation.

$$\frac{d}{dw}\left[(xw - y)^2\right]$$

$$= 2(xw - y) \cdot \frac{d}{dw}[xw - y]$$

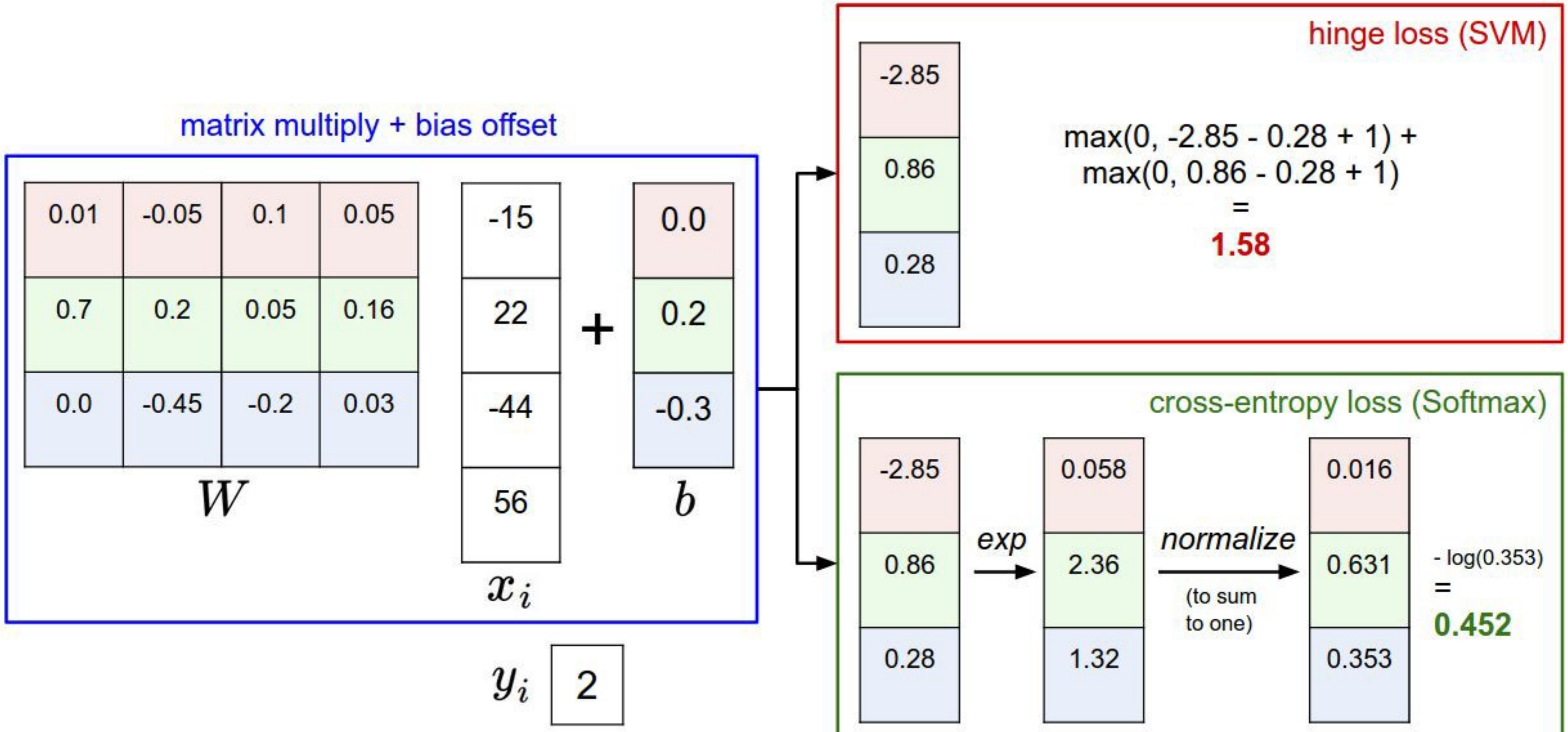
$$= 2\left(x \cdot \frac{d}{dw}[w] + \frac{d}{dw}[-y]\right)(xw - y)$$

$$= 2(1x + 0)(xw - y)$$

$$= 2x(xw - y)$$

Table of contents

Loss Function to calculate the errors



Loss Function (Cross-Entropy and Hinge (SVM Loss))

Cross-Entropy-Loss (after Softmax)

Let (x_i, y_i) be a pair of training example, where $x_i \in R^D$ is training data and $y_i \in R^K$ is a one-hot vector with all zeros except for its true class index is one. Assume the prediction $\hat{y}_i \in R^K$ is computed after Softmax, then the cross-entropy loss is:

$$L_i = -y_i \cdot \log(\hat{y})$$

Hinge Loss (SVM Loss)

Note that the exponential computing in cross-entropy loss is expensive and sometimes we do not need a probability. Thus, the hinge loss function is:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Table of contents

Softmax Classifier (Multinomial Logistic Regression)

cat 3.2

car 5.1

frog -1.7

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s=f(x_i;W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i=-\log P(Y=y_i|X=x_i)$$

in summary: $$L_i=-\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Softmax Classifier

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat	**3.2**
car	5.1
frog	-1.7

unnormalized log probabilities

Softmax Classifier

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**		**24.5**
car	5.1	exp →	164.0
frog	-1.7		0.18

unnormalized log probabilities

Softmax Classifier

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	3.2		24.5		0.13
car	5.1	exp →	164.0	normalize →	0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

Softmax + Cross entropy

$$L_i = -y_i \cdot \log(\hat{y})$$

unnormalized probabilities

	unnormalized log probabilities		unnormalized probabilities		probabilities
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

Table of contents

Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Table of contents

Overall Loss

The overall loss consists of two items, namely data loss and regularization loss.

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W)$$

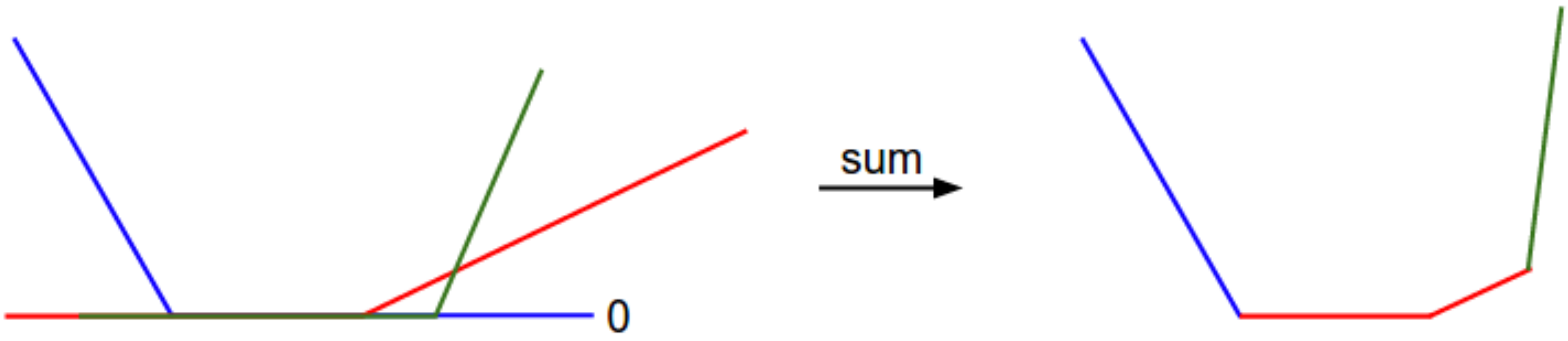


Table of contents

Summary of loss function

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss

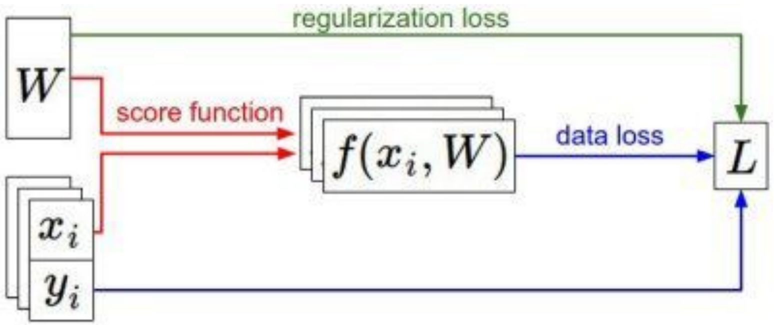


Table of contents

Follow the slope

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives).

Follow the slope

current W:	**W + h** (first dim)**:**	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[?,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

Follow the slope

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim):

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[**-2.5**,
?,
?,
?,
?,…]

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Gradient Decent

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Gradient Decent

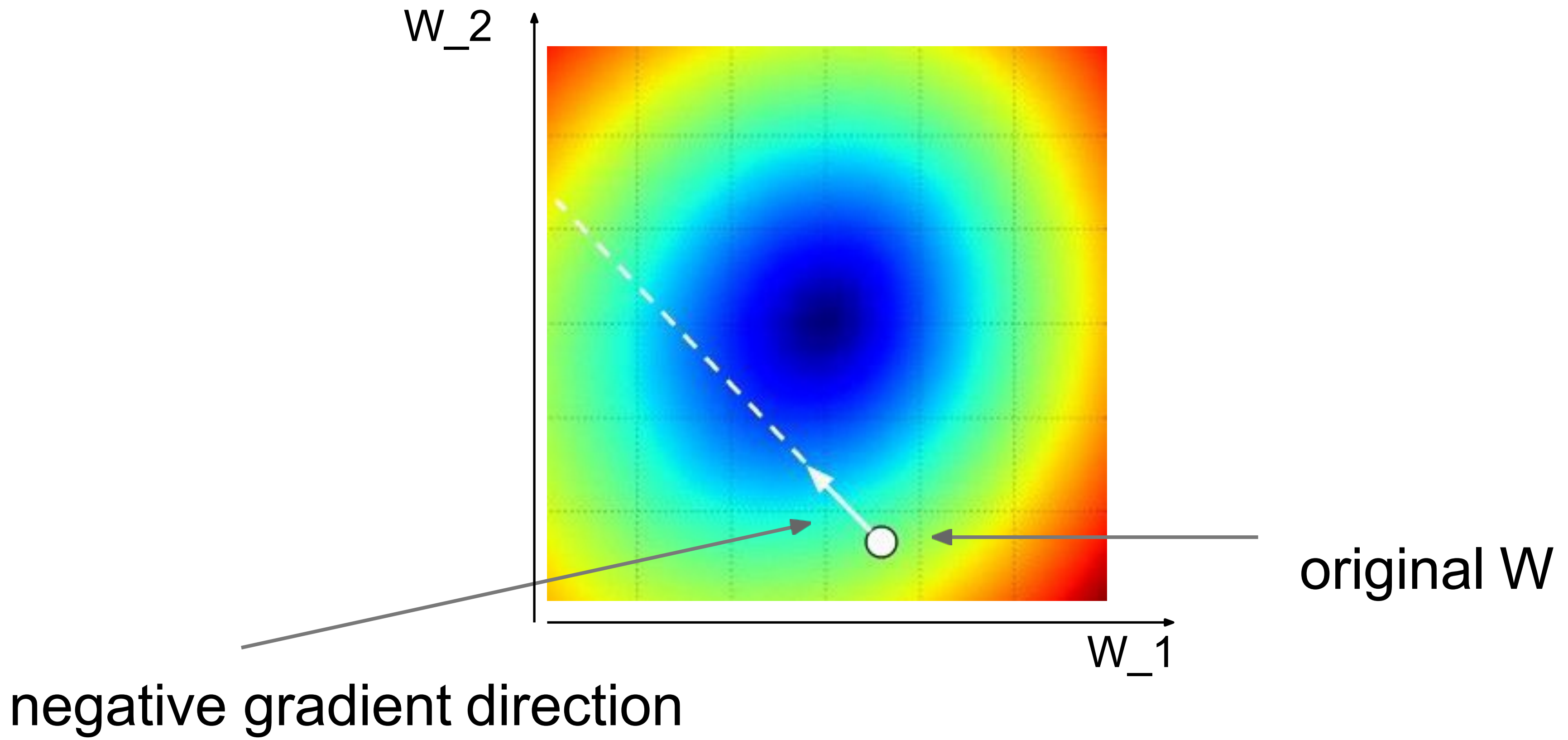
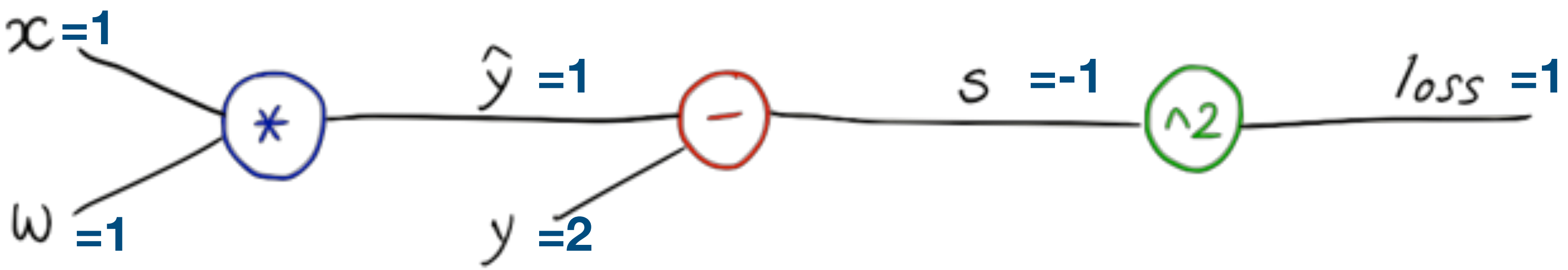


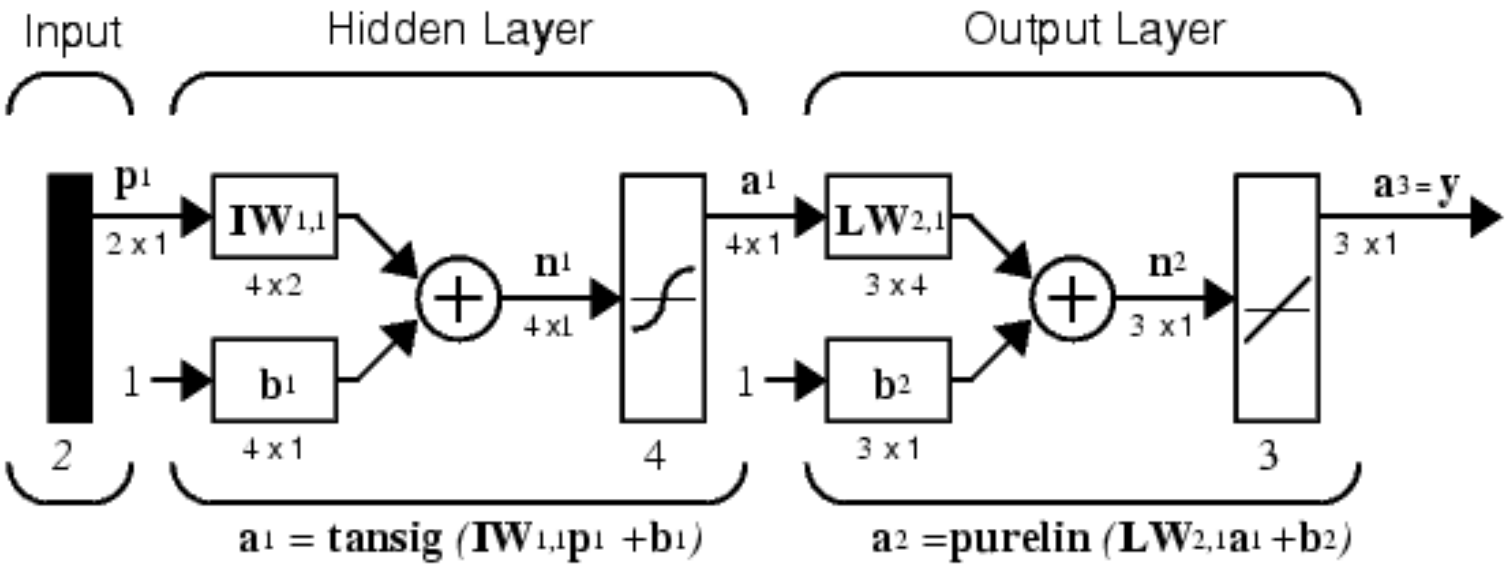
Table of contents

Feed forwarding



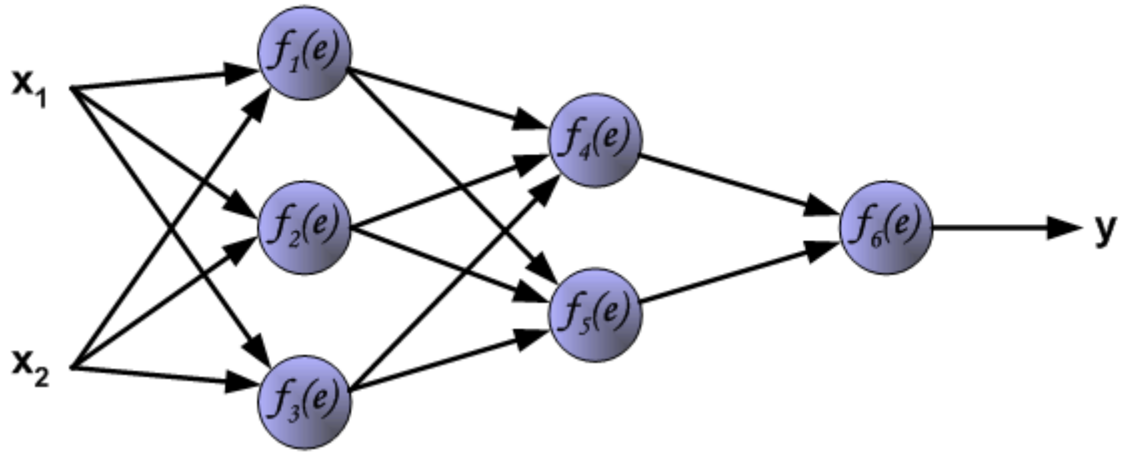
Feed forwarding

Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons.
Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors.
The linear output layer lets the network produce values outside the range -1 to +1.
On the other hand, if you want to constrain the outputs of a network (such as between 0 and 1), then the output layer should use a sigmoid transfer function (such as logsig).

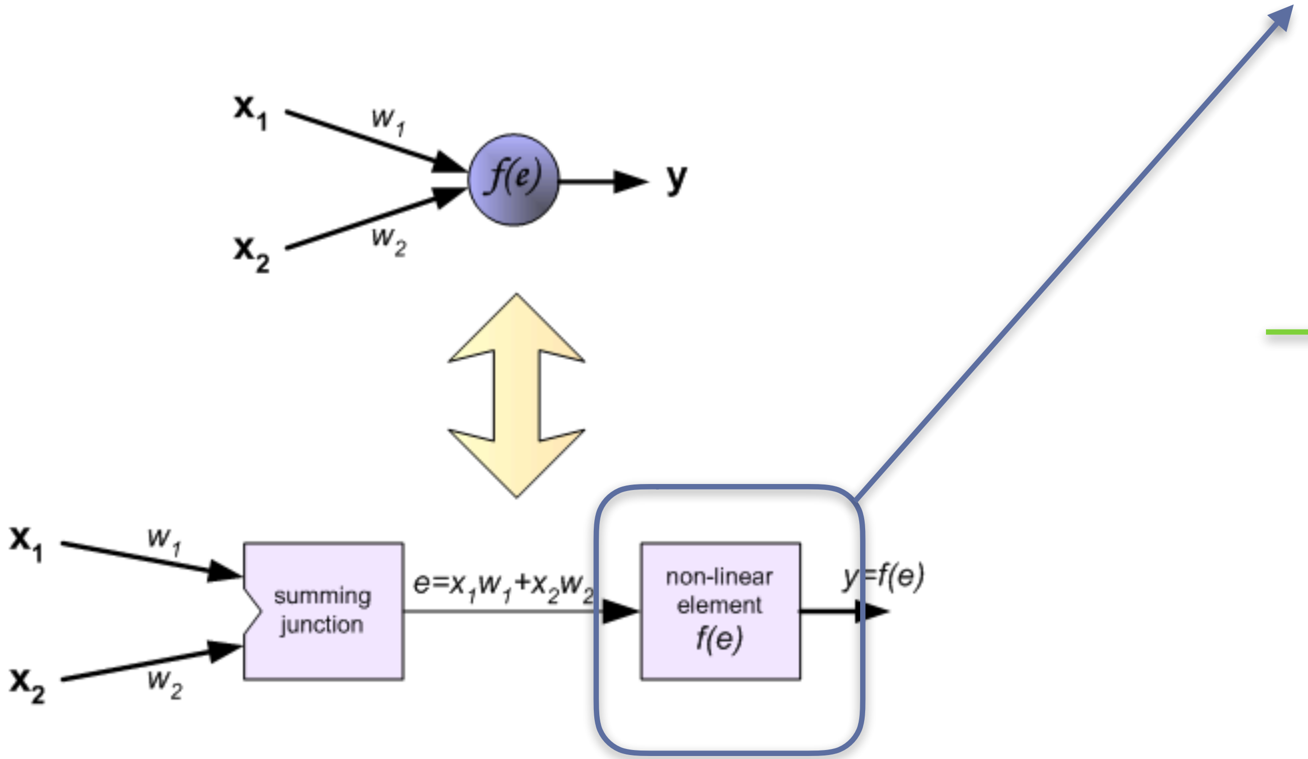


Feed forwarding

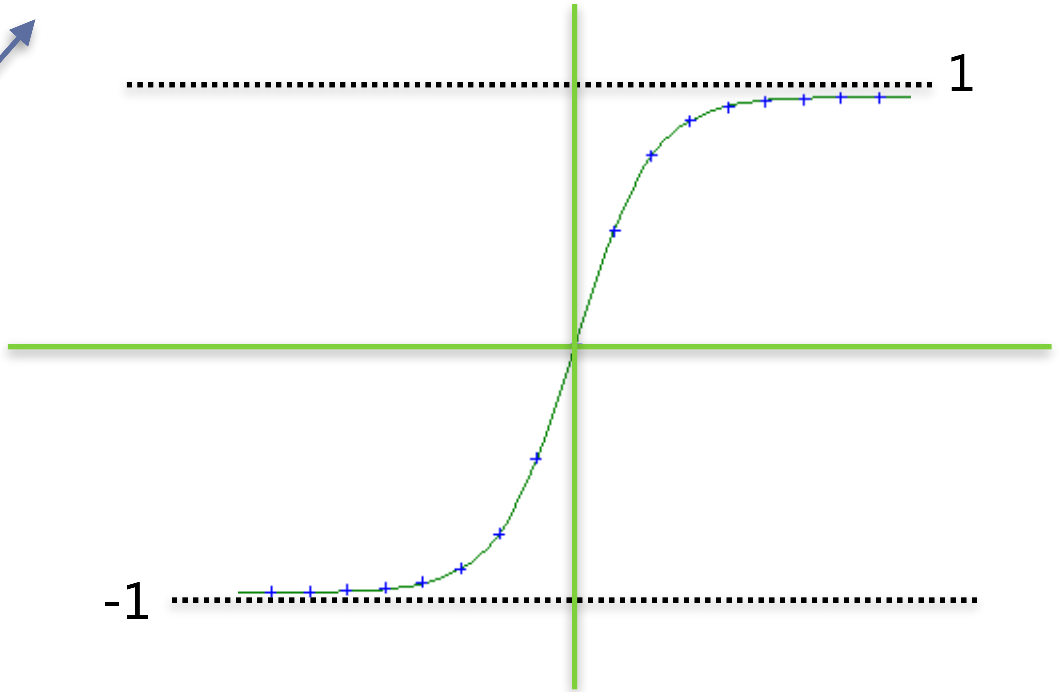
The following slides describes **teaching process** of multi-layer neural network employing **backpropagation** algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:



Active Function



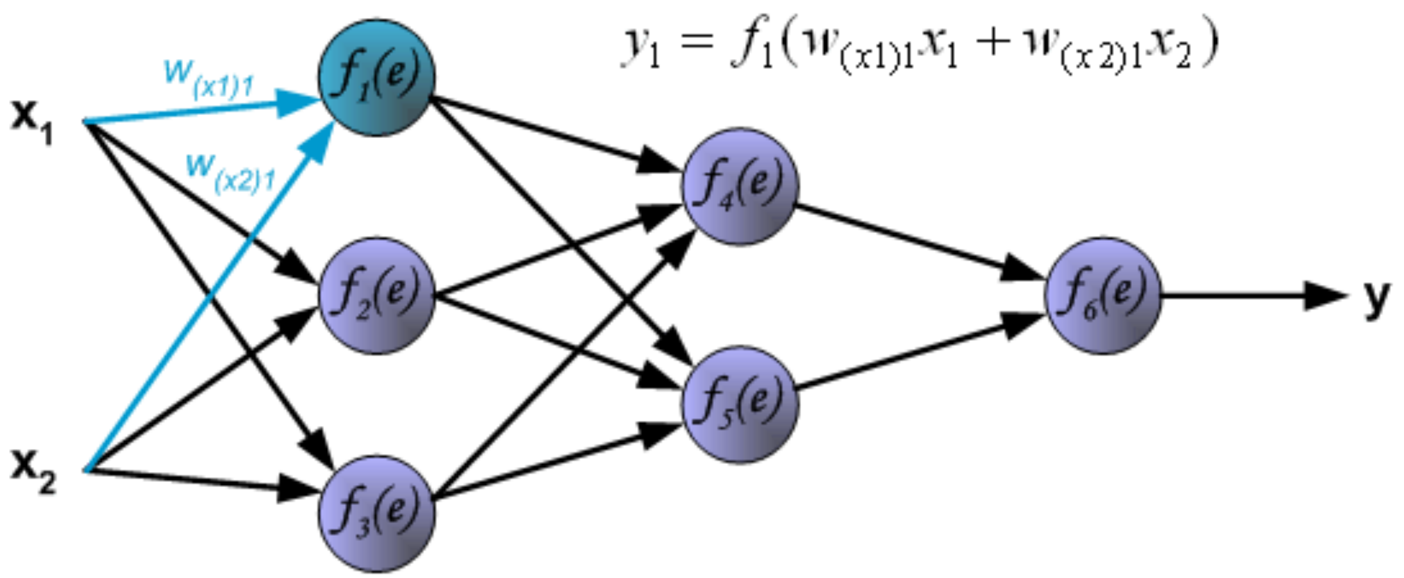
Active Function: Sigmoid



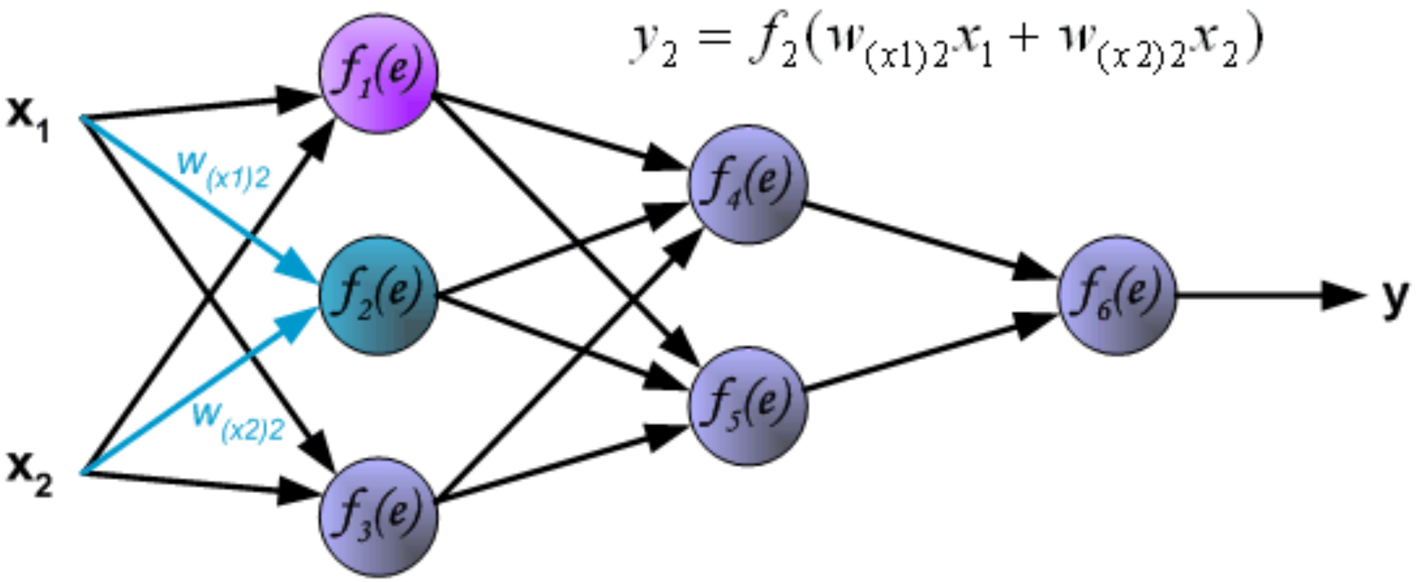
$$f(x)=\frac{1}{1+e^{-\lambda x}}$$

Feed forwarding

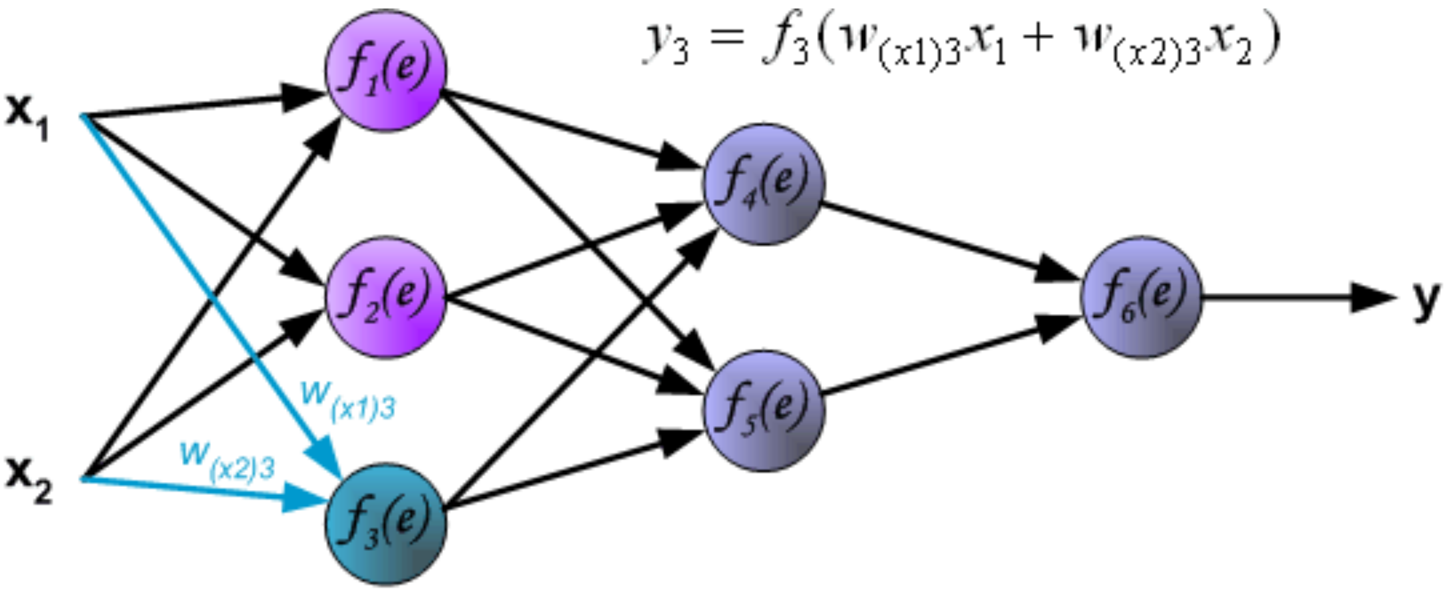
Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



Feed forwarding

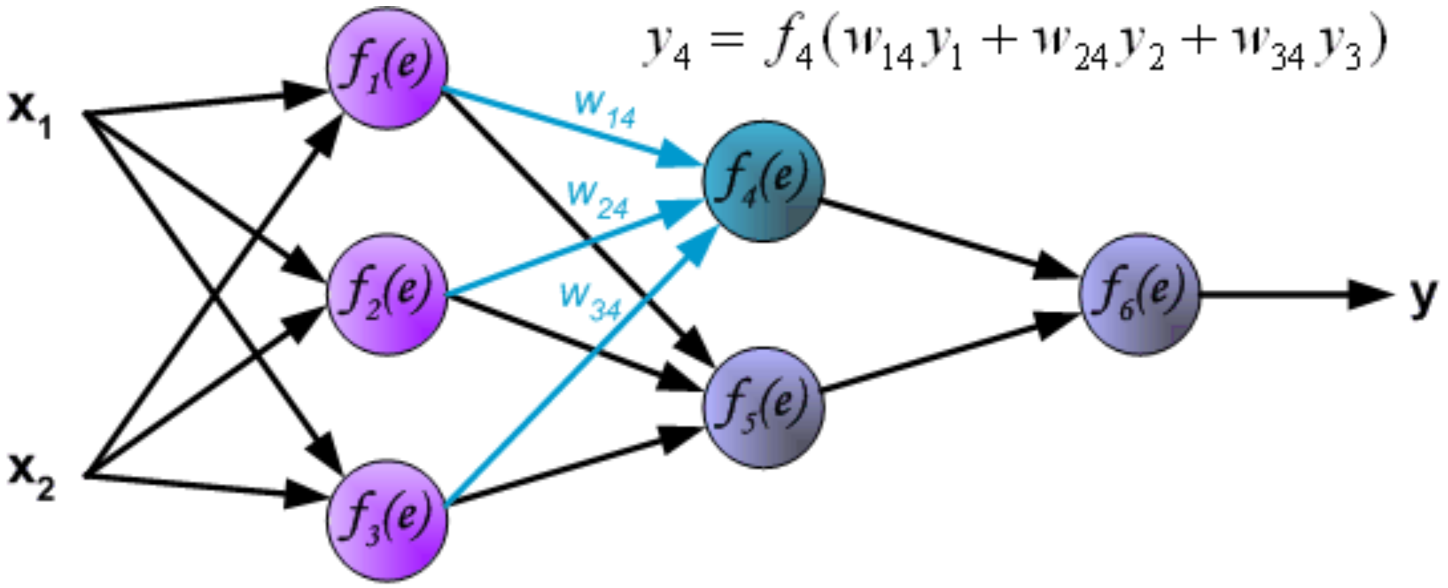


Feed forwarding

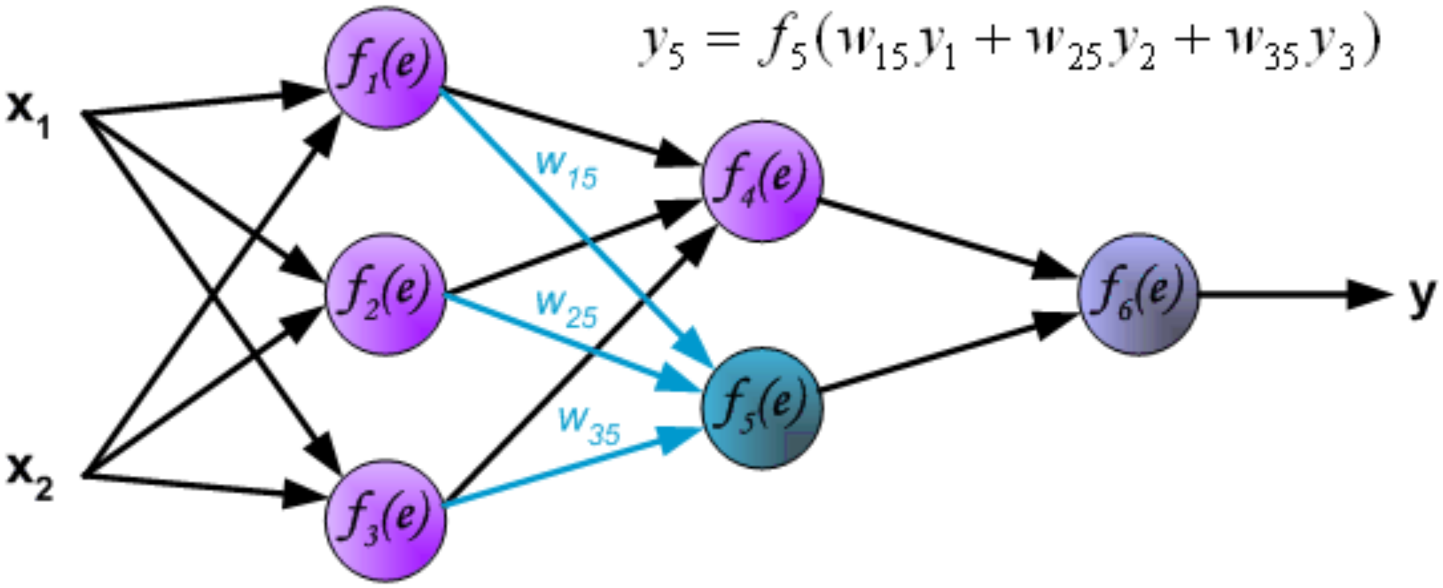


Feed forwarding

Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.



Feed forwarding



Feed forwarding

Propagation of signals through the output layer.

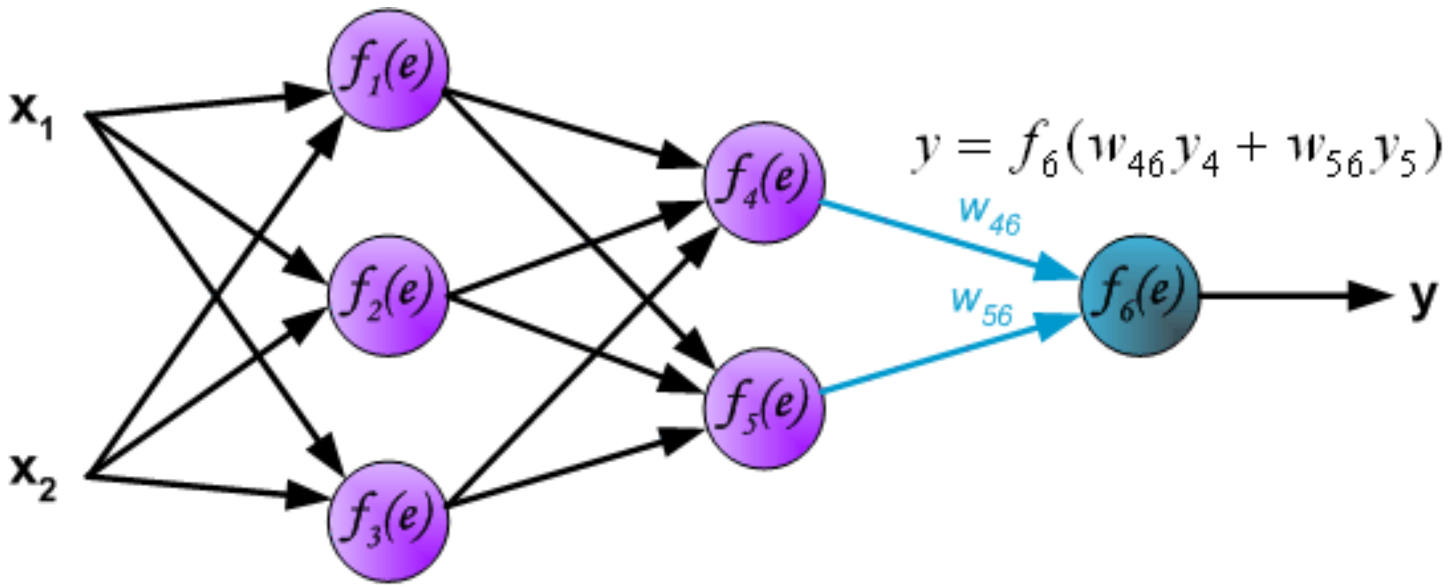
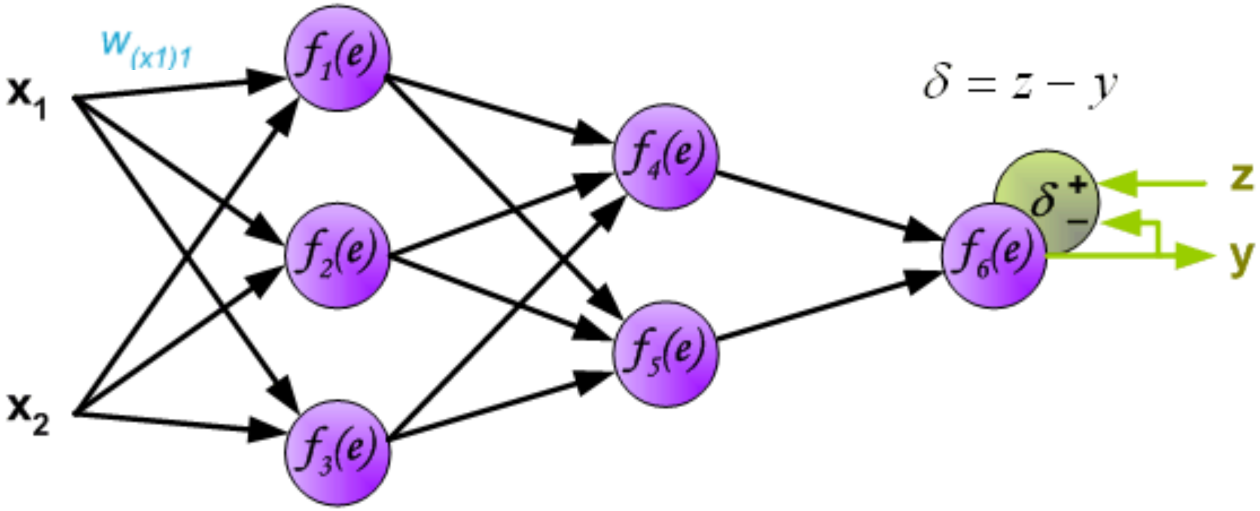


Table of contents

Back Propogation

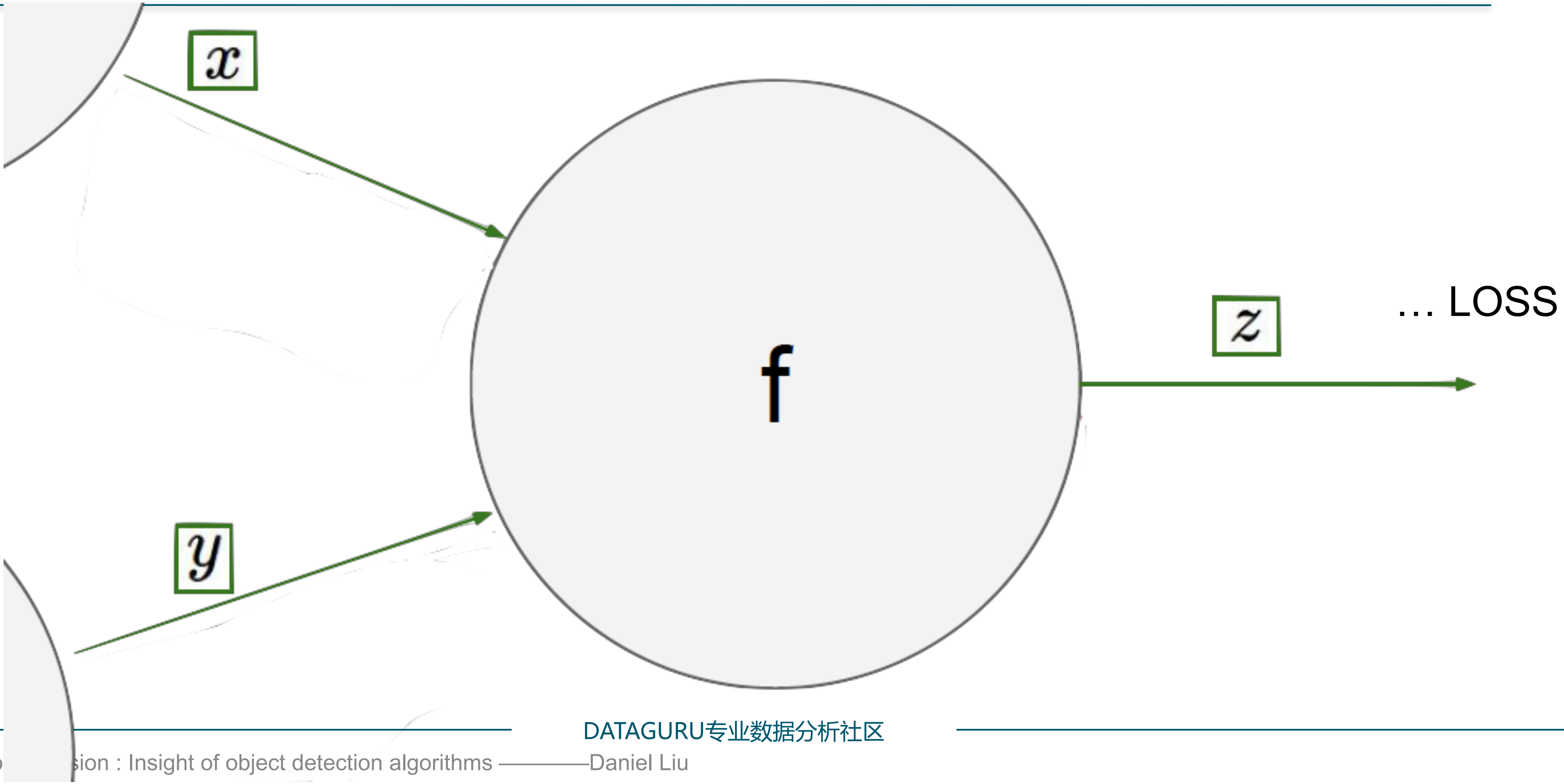


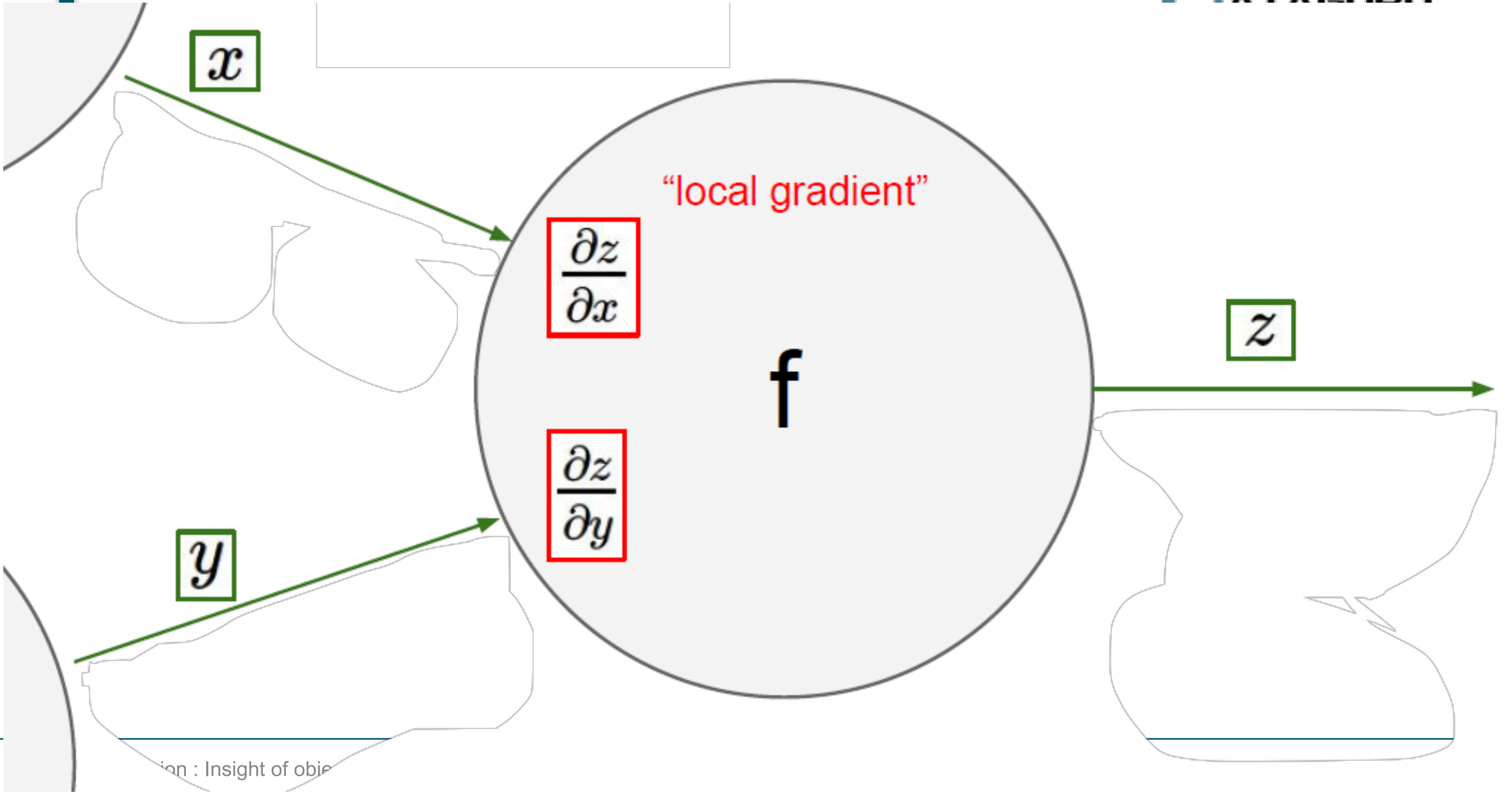
Gradient of **loss** with respect to ***w*** $\frac{\partial loss}{\partial w} = ?$

$$\frac{\partial \delta}{\partial W} = ?$$

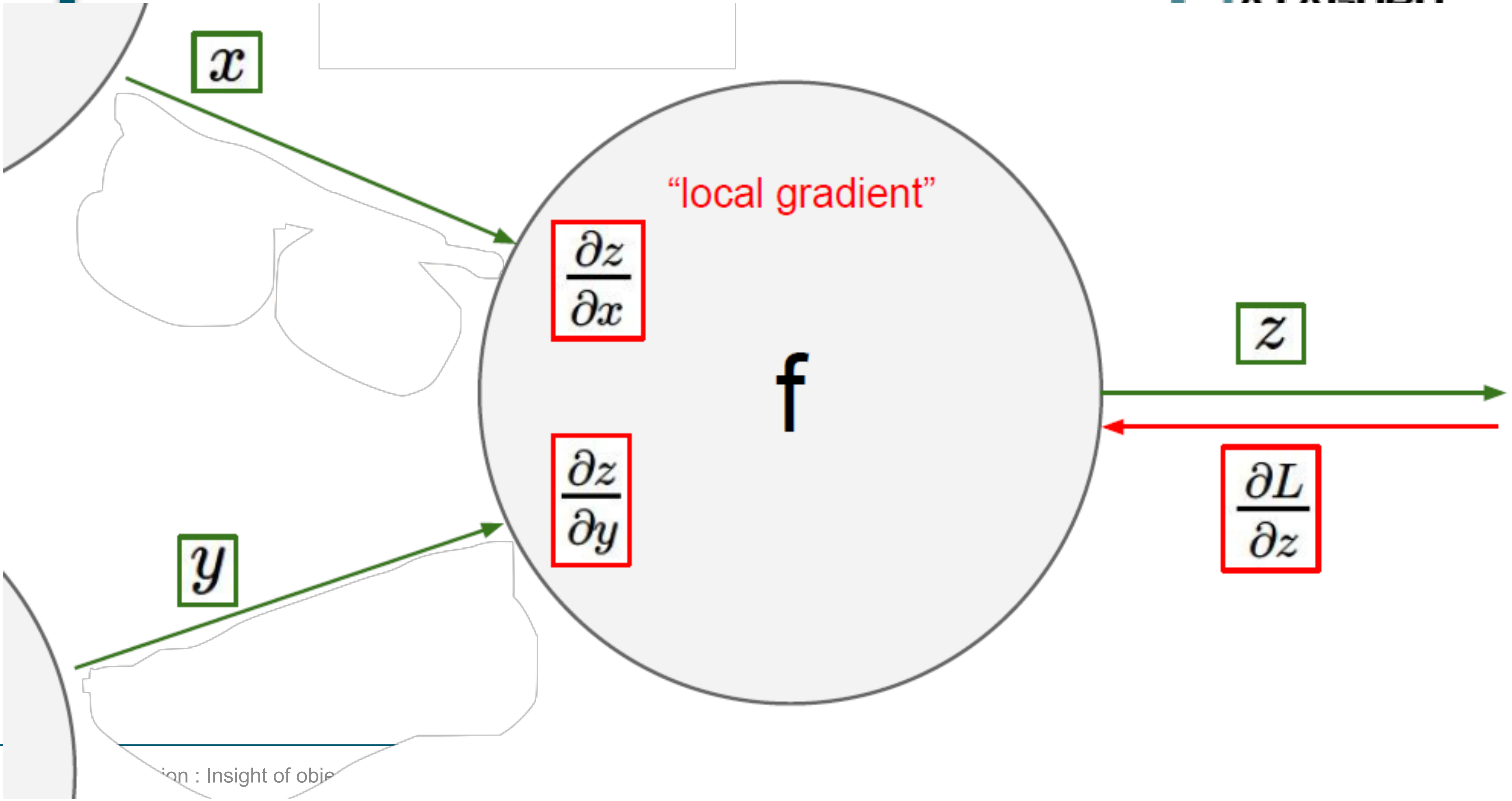
Table of contents

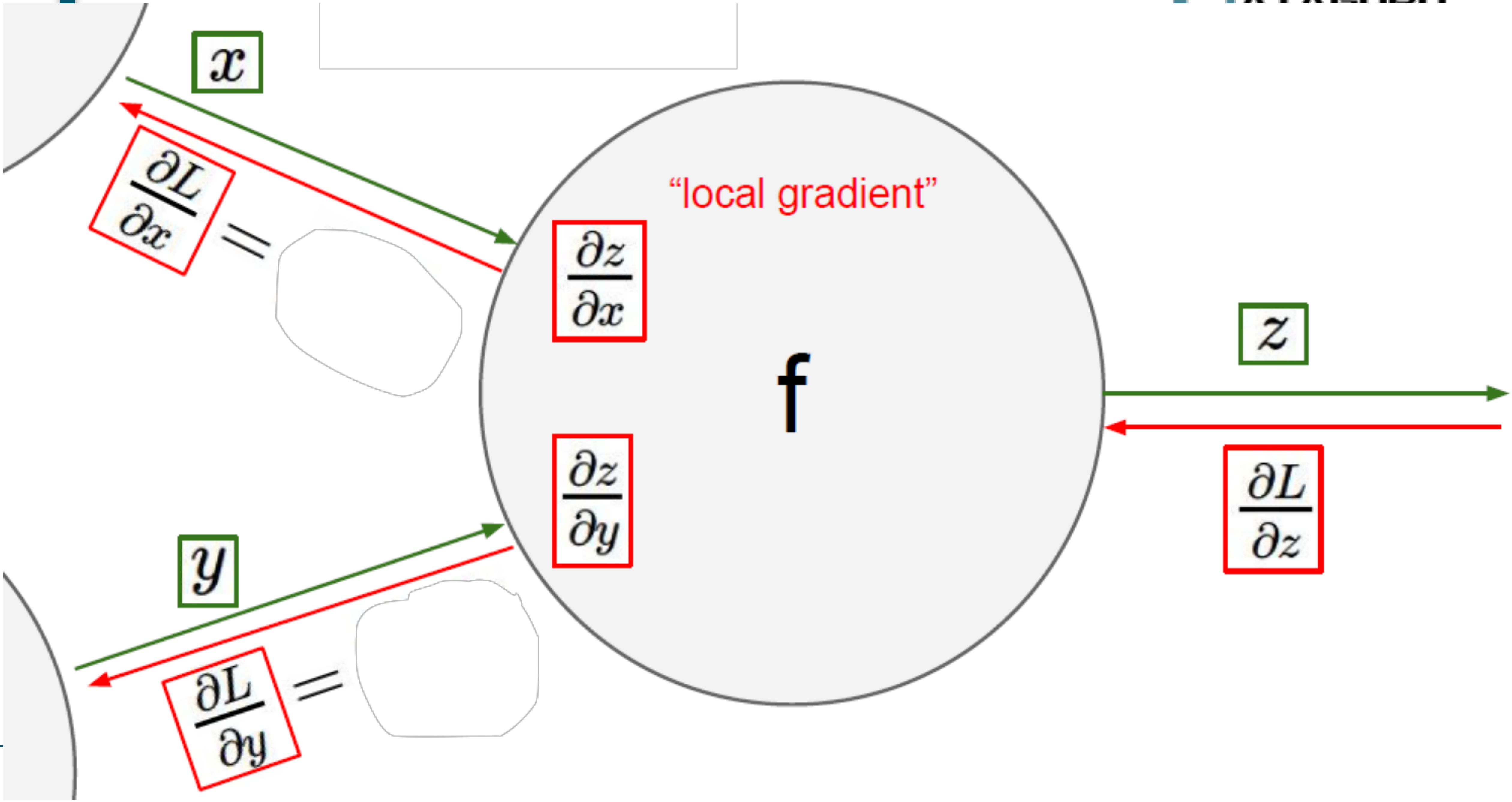
Chain rule



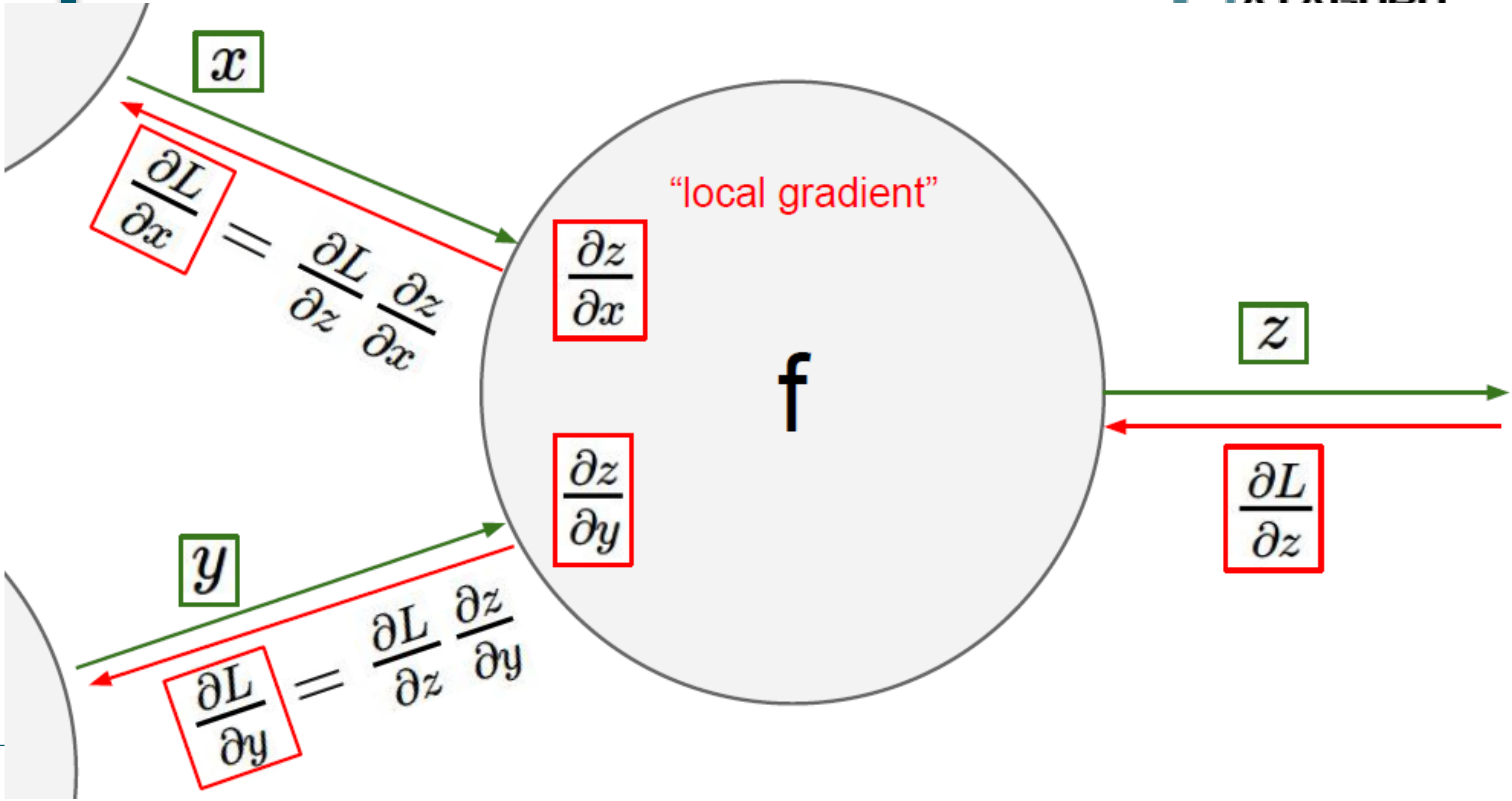


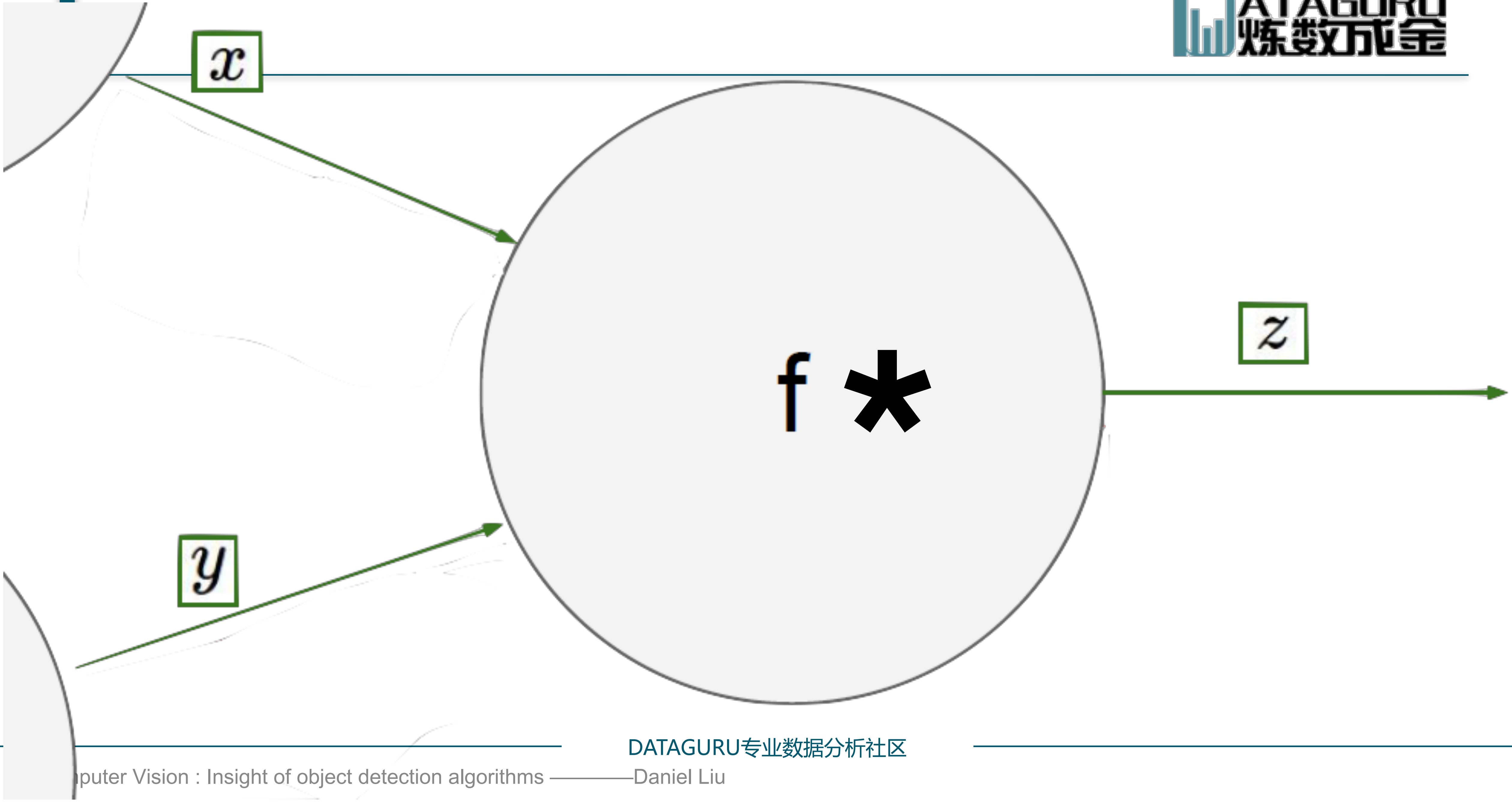
x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y


x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$

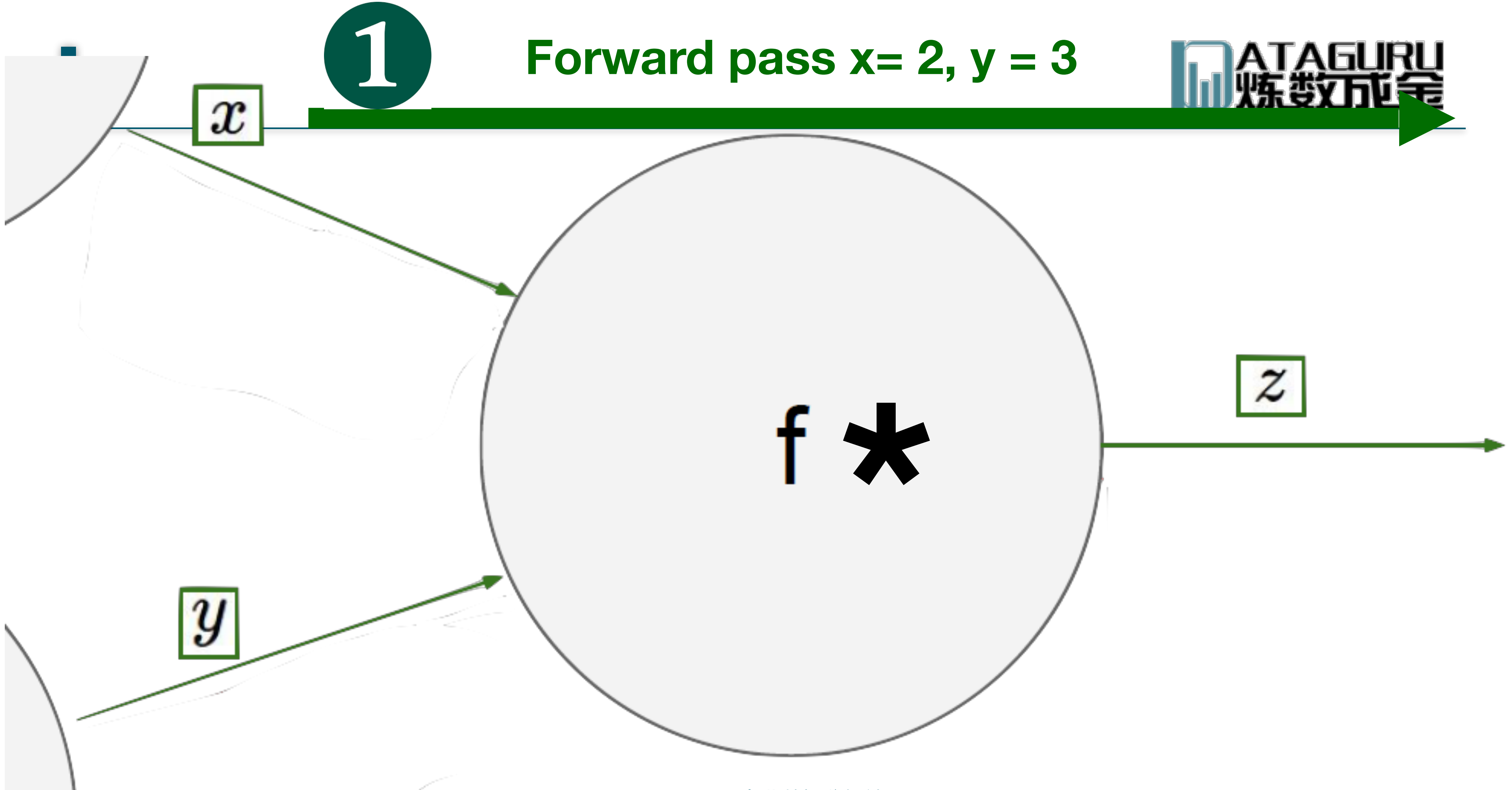


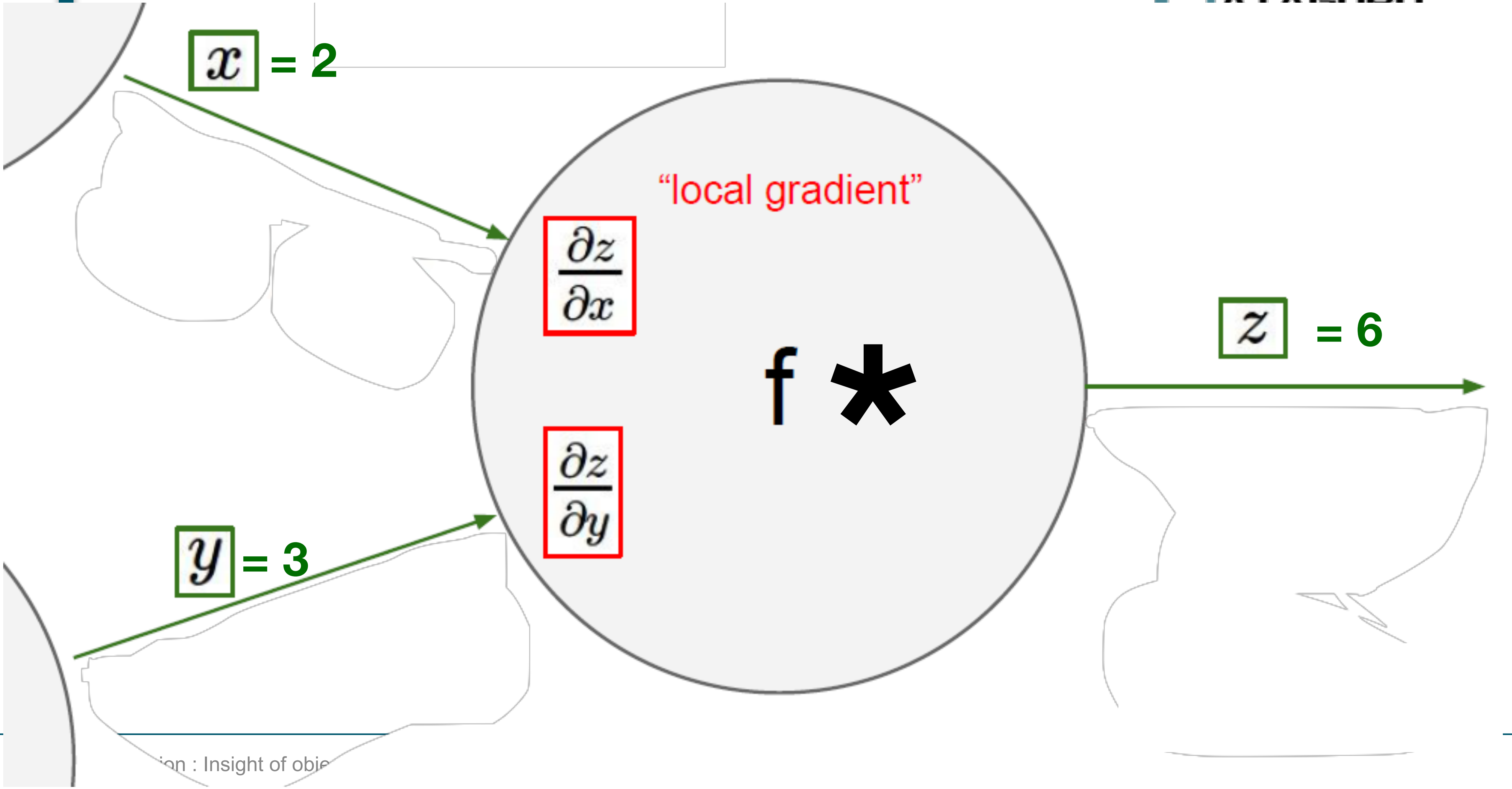
x
$\frac{\partial L}{\partial x} =$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} =$


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$


x
y
f *
z

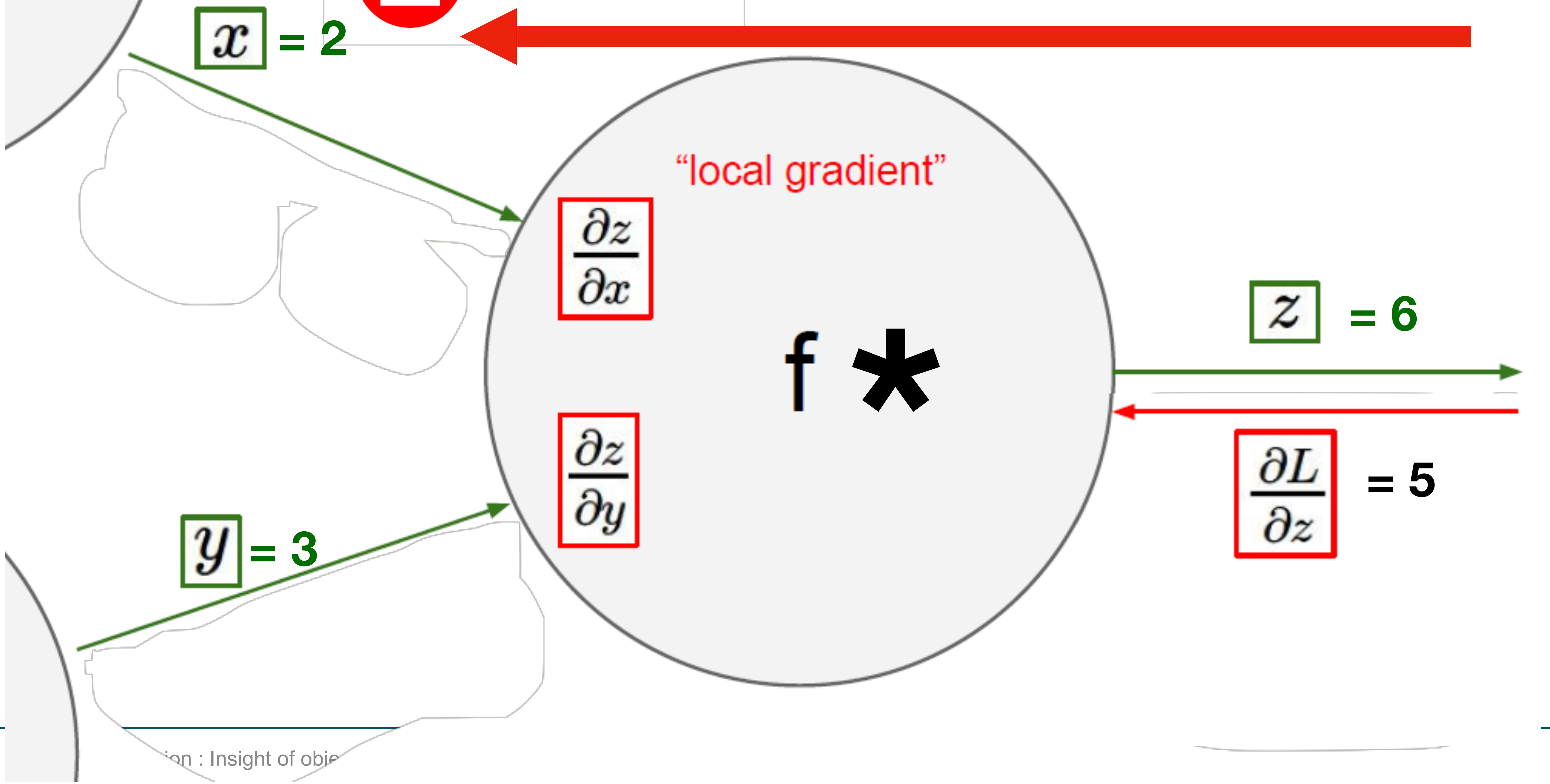
Forward pass x= 2, y = 3

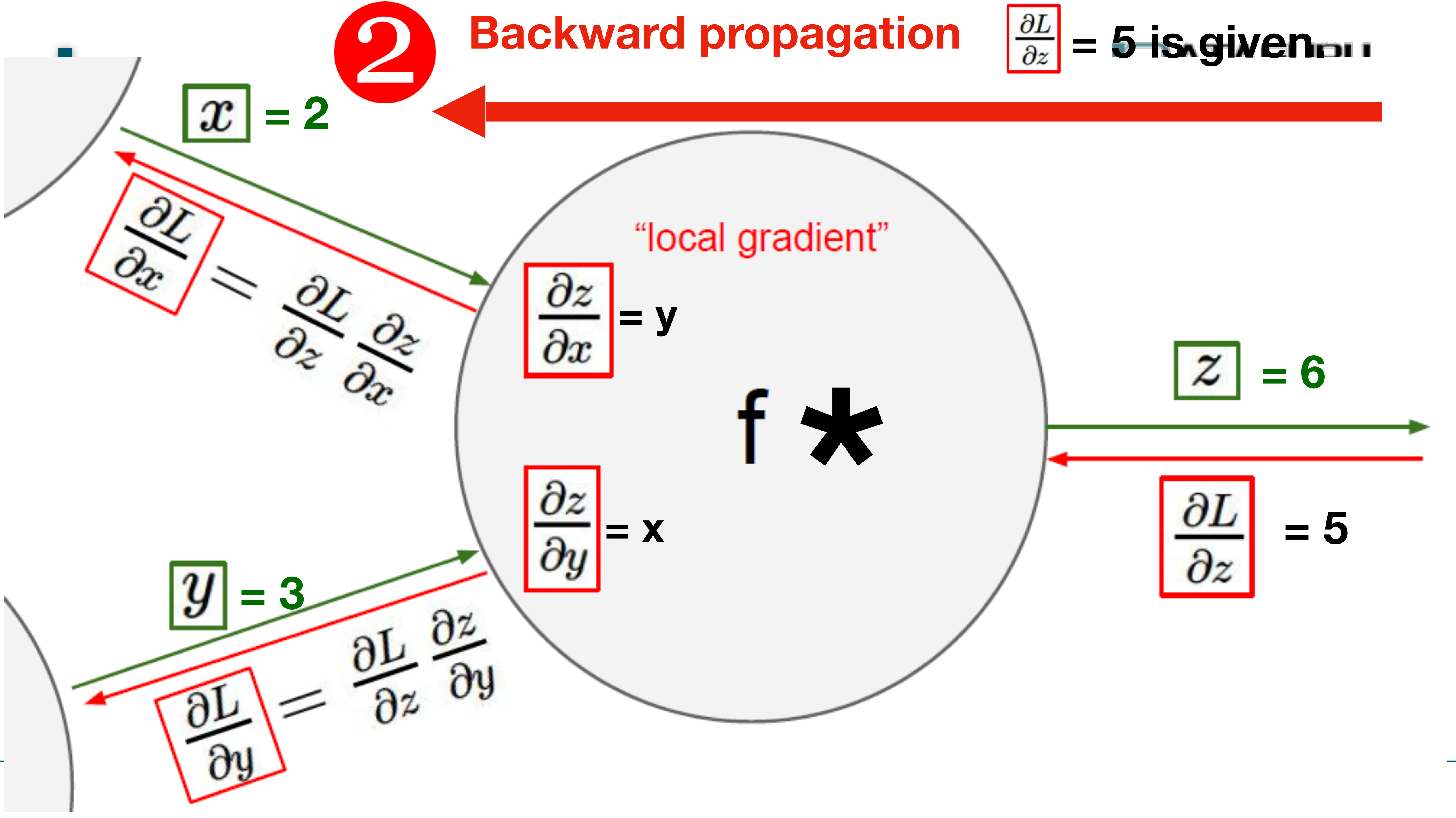




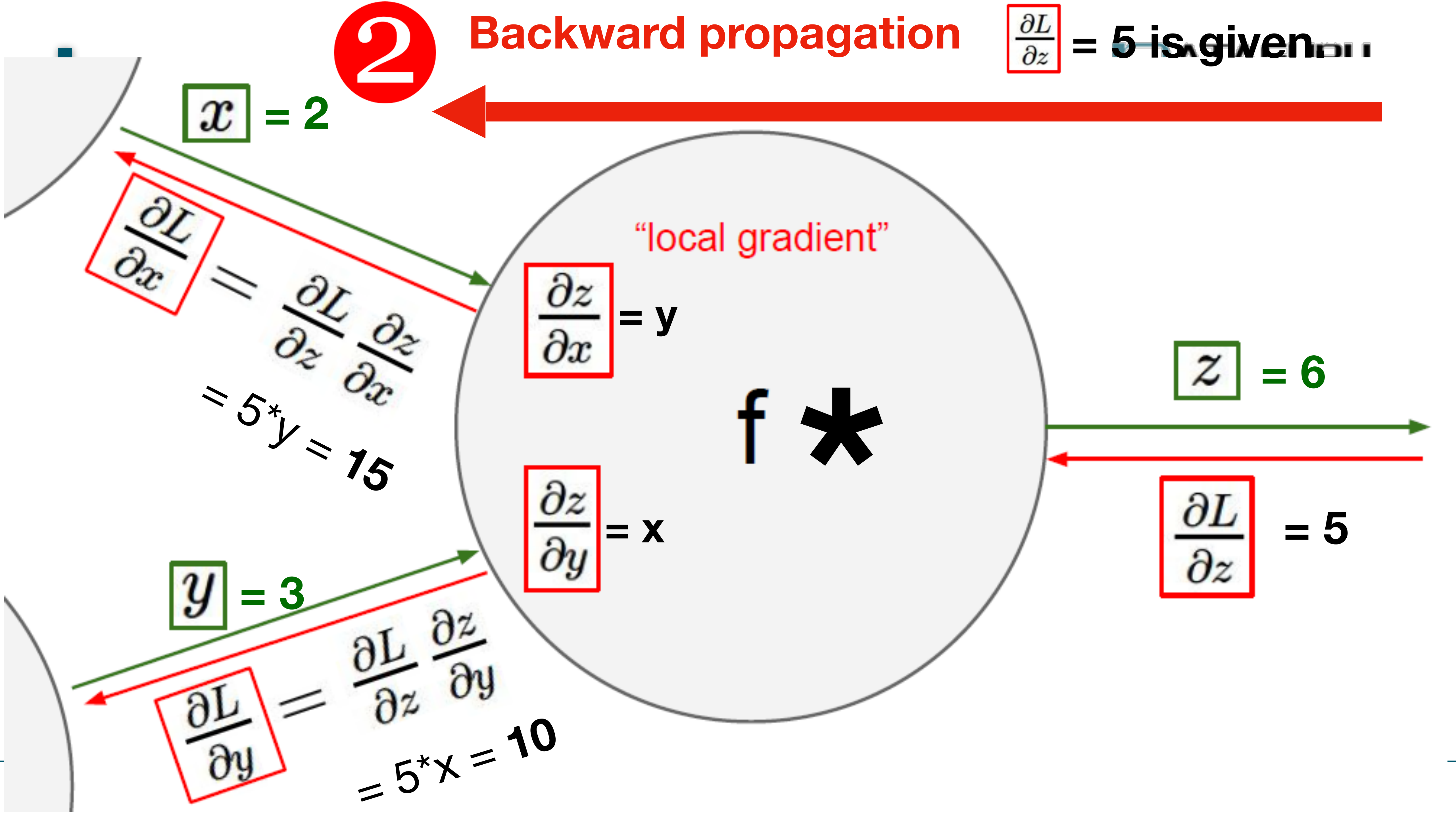
x = 2
"local gradient"
$\frac{\partial z}{\partial x}$
f *
z = 6
$\frac{\partial z}{\partial y}$
y = 3

❷ Backward propagation $\frac{\partial L}{\partial z}$ = 5 is given





2
Backward propagation
∂L/∂z = 5 is given
x = 2
y = 3
z = 6
∂L/∂x = ∂L/∂z ∂z/∂x
∂L/∂y = ∂L/∂z ∂z/∂y
"local gradient"
∂z/∂x = y
∂z/∂y = x
f *
∂L/∂z = 5



2
Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given
x = 2
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
= 5*y = 15
"local gradient"
$\frac{\partial z}{\partial x}$ = y
f *
$\frac{\partial z}{\partial y}$ = x
z = 6
$\frac{\partial L}{\partial z}$ = 5
y = 3
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
= 5*x = 10

Table of contents

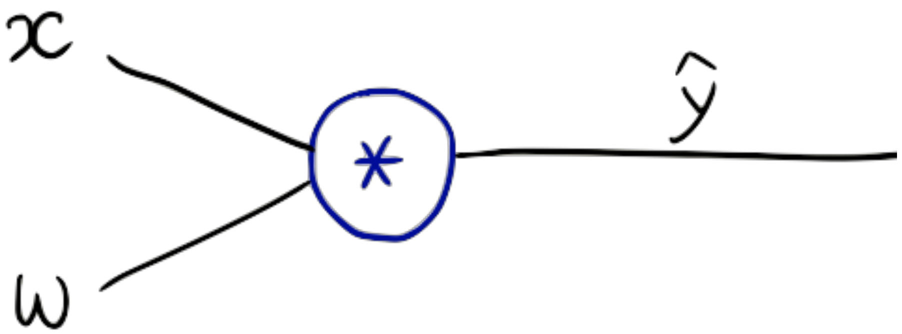
Computational graph

$$\hat{y} = x * w$$

Computational graph

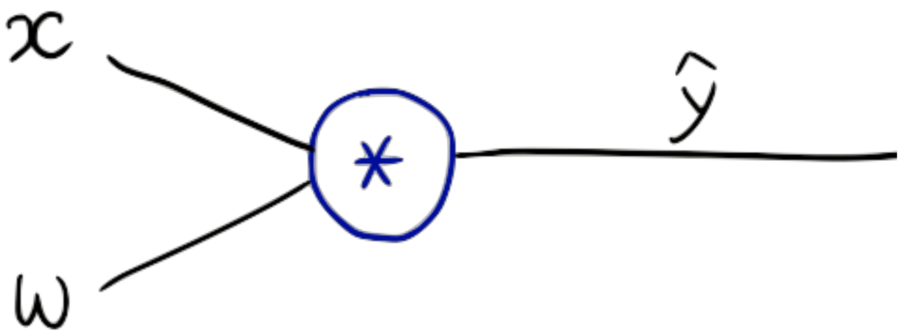
DATAGURU 炼数成金

$$\hat{y} = x * w$$



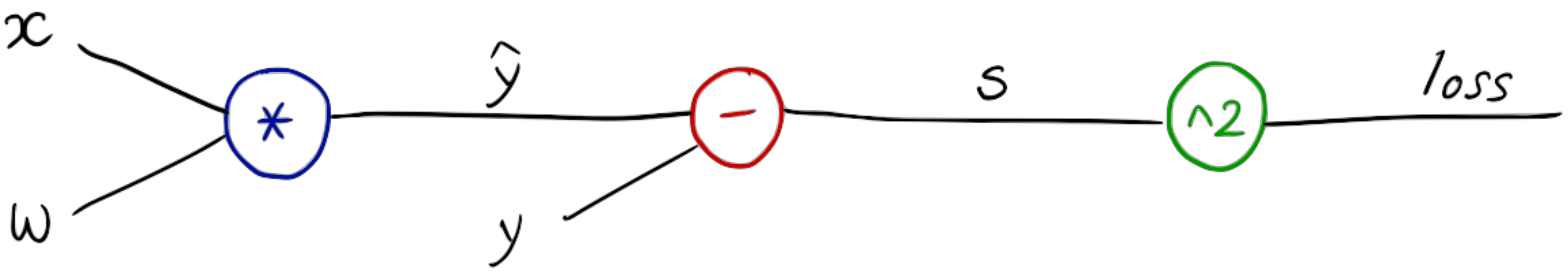
Computational graph

$$\hat{y} = x * w \qquad loss = (\hat{y} - y)^2 = (x * w - y)^2$$

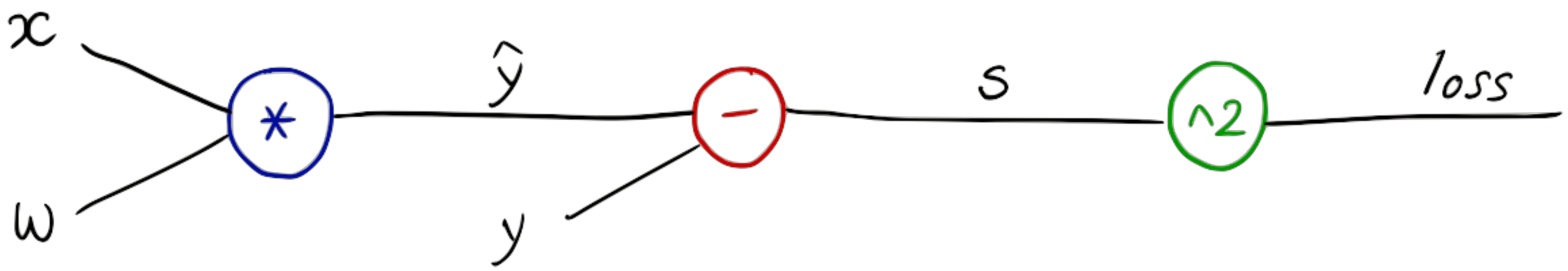


Computational graph

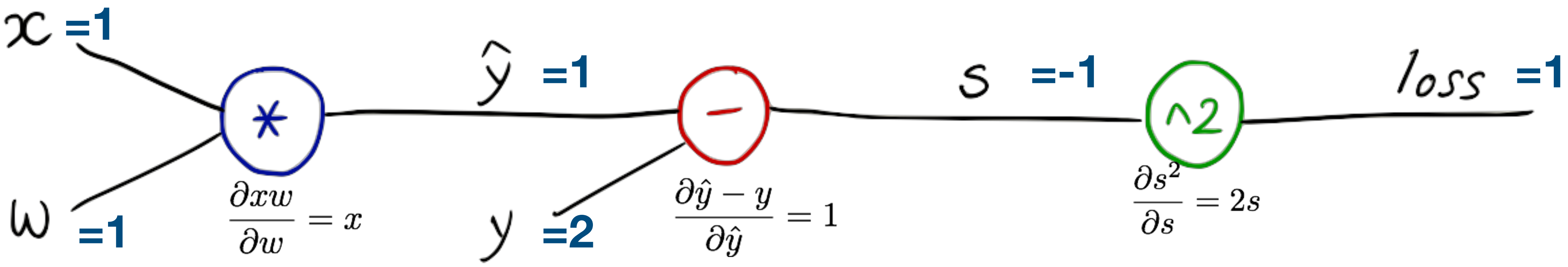
$$\hat{y} = x * w \qquad loss = (\hat{y} - y)^2 = (x * w - y)^2$$



1 Forward pass x=1, y = 2 where w=1

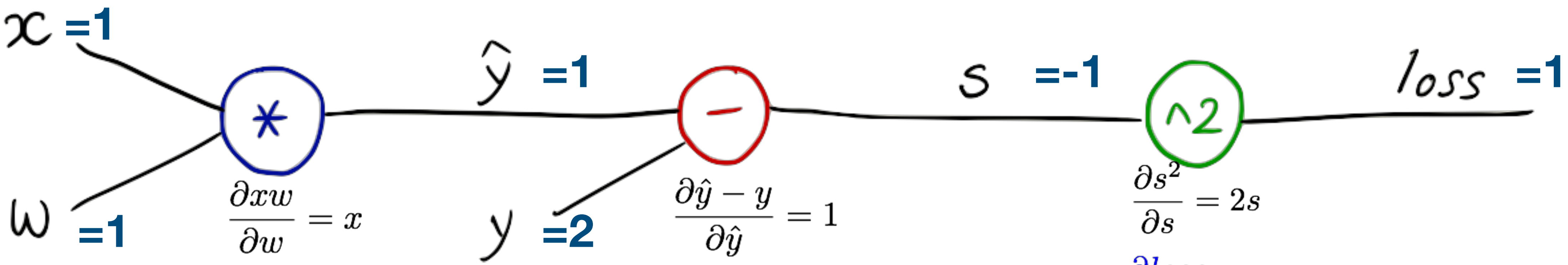


2 Backward propagation



$$\frac{\partial loss}{\partial w} =$$

❷ Backward propagation



$$\frac{\partial xw}{\partial w} = x$$

$$\frac{\partial \hat{y} - y}{\partial \hat{y}} = 1$$

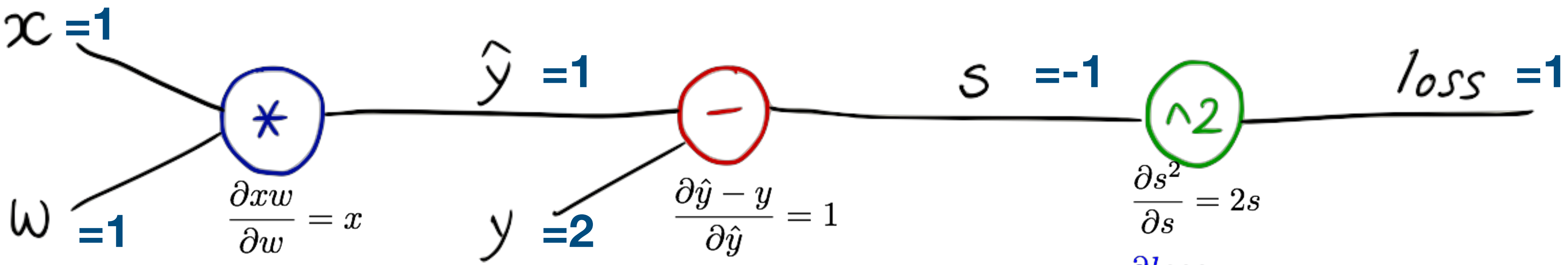
$$\frac{\partial s^2}{\partial s} = 2s$$

$$\frac{\partial loss}{\partial s} = 2s = -2$$

$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s}\frac{\partial s}{\partial \hat{y}} = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2 * x = -2 * 1 = -2$$

② Backward propagation



$$\frac{\partial xw}{\partial w} = x$$

$$\frac{\partial \hat{y} - y}{\partial \hat{y}} = 1$$

$$\frac{\partial s^2}{\partial s} = 2s$$

$$\frac{\partial loss}{\partial s} = 2s = -2$$

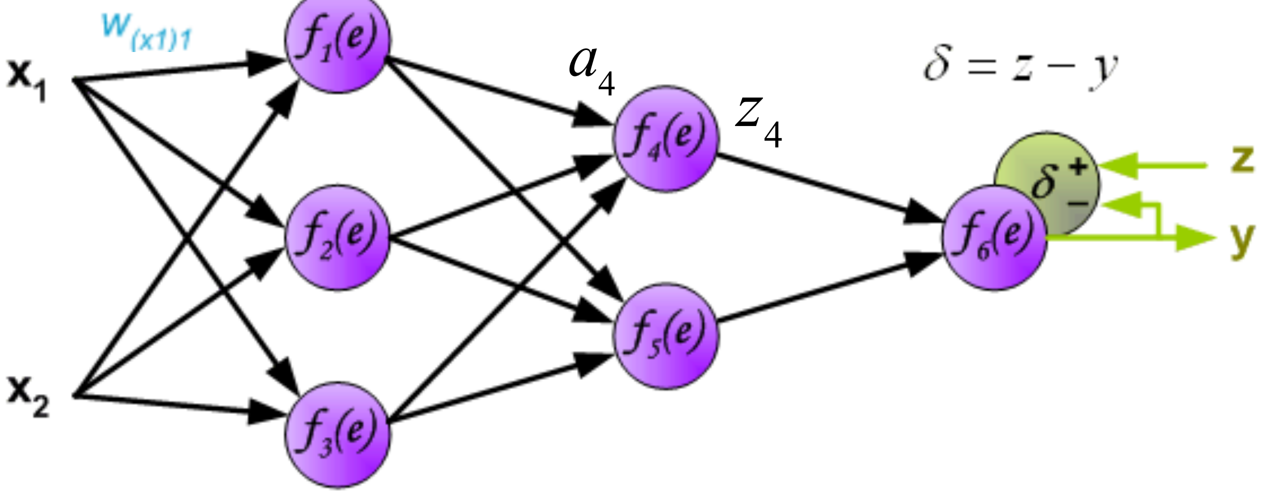
$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s}\frac{\partial s}{\partial \hat{y}} = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2 * x = -2 * 1 = -2$$

```
grad:  1.0 2.0 -2.0
grad:  2.0 4.0 -7.84
grad:  3.0 6.0 -16.2288
progress: 0 4.919240100095999
```

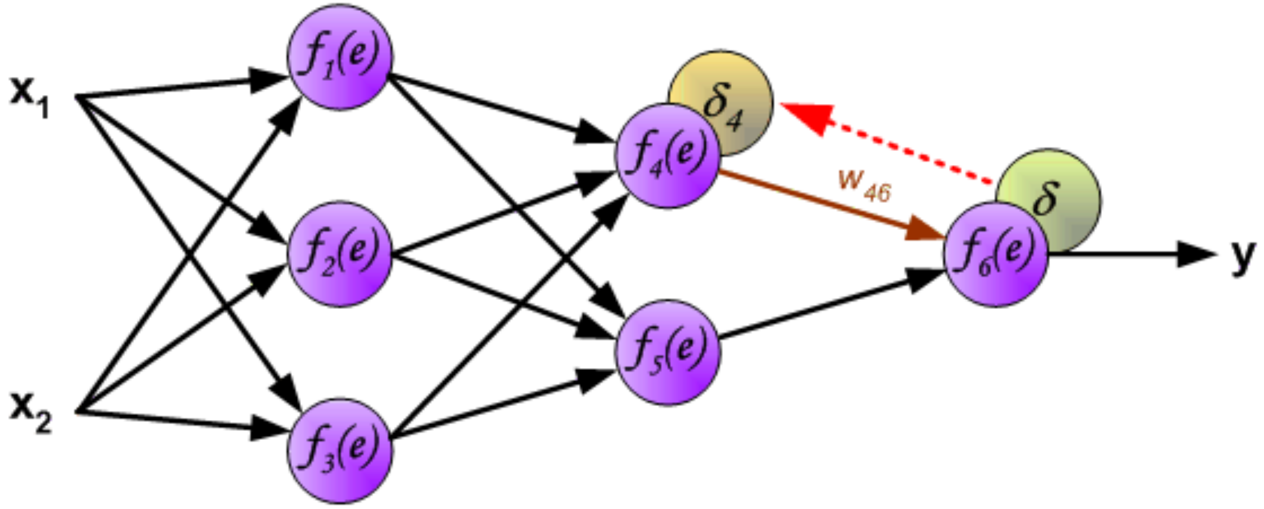
Table of contents

Backward propagation



$$\frac{\partial \delta}{\partial z_4} = ?$$

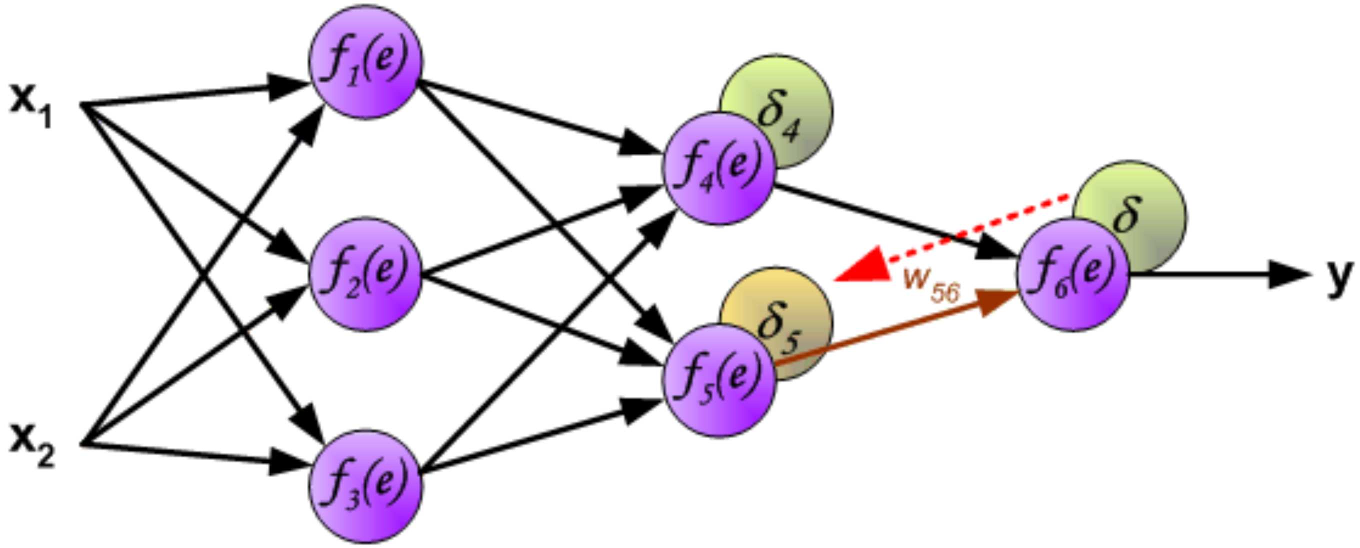
Backward propagation to the 2nd layer



$$\delta_4 = \delta \frac{\partial \delta}{\partial z_4} = \delta \frac{\partial \left(W_{46} z_4 + b_{46} \right)}{\partial z_4} \cdot f_6'$$

$$= \delta W_{46} f_6'$$

Backward propagation to the 2nd layer

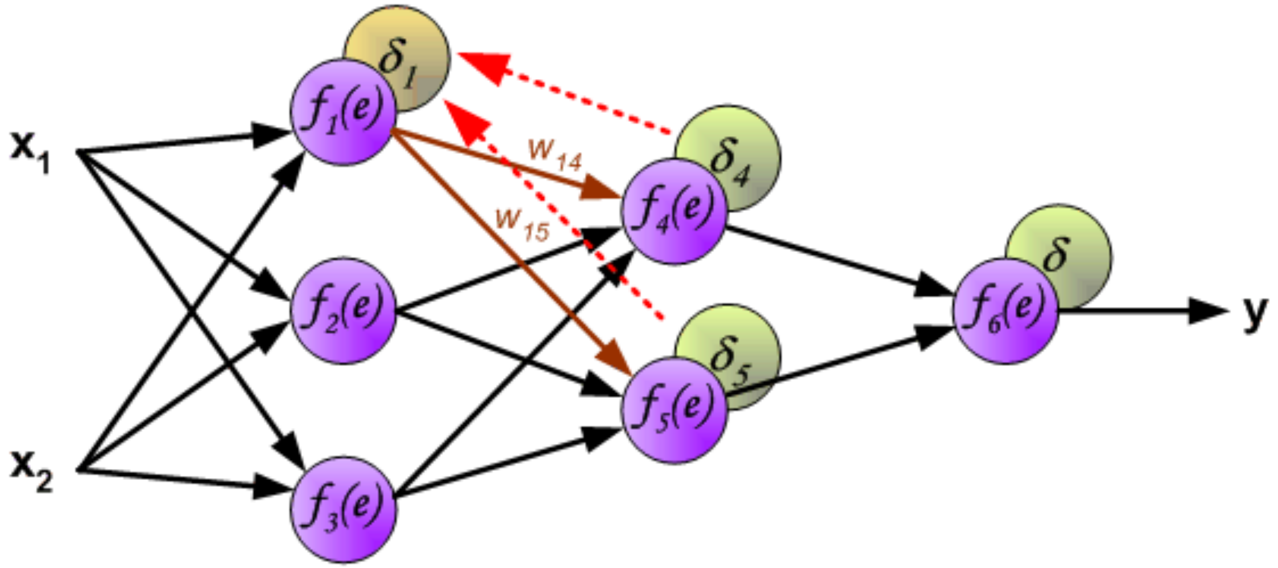


$$\delta_4 = \delta \frac{\partial \delta}{\partial z_4} = \delta \frac{\partial \left(W_{46} z_4 + b_{46} \right)}{\partial z_4} \cdot f_6'$$

$$= \delta W_{46} f_6'$$

$$\delta_5 = \delta W_{56} f_6'$$

Backward propagation to the 1st layer



$$\delta_1 = \delta_4 W_{14} f_4' + \delta_5 W_{15} f_5'$$

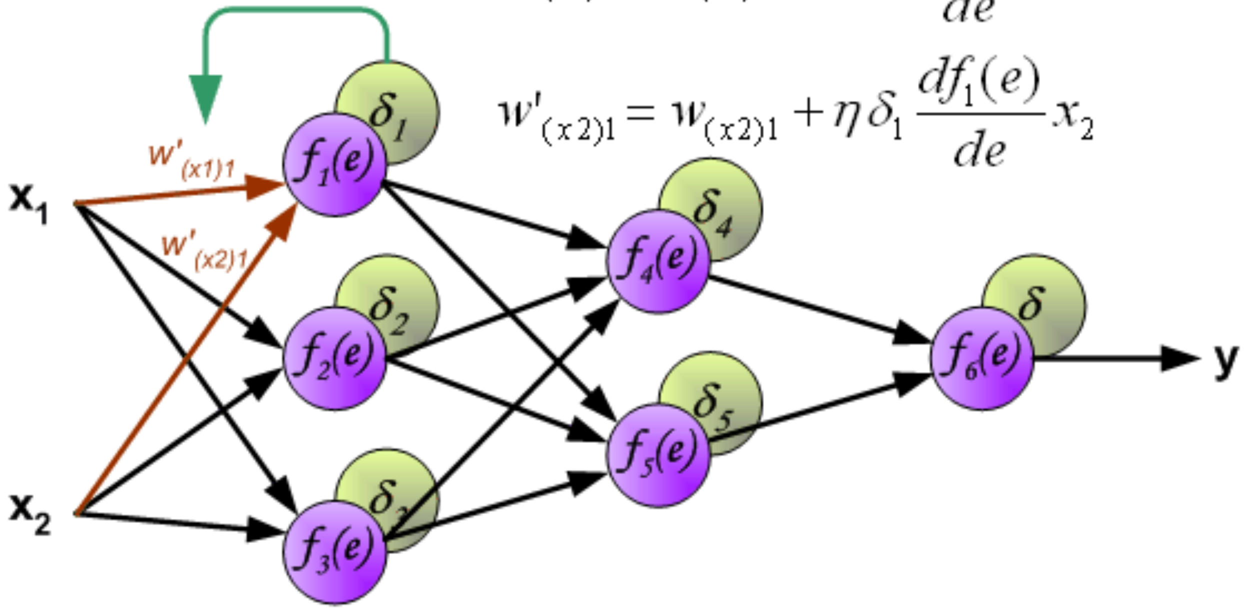
$$= \delta f_6' (W_{14} W_{46} f_4' + W_{56} W_{15} f_5')$$

1st Layer Refresh

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

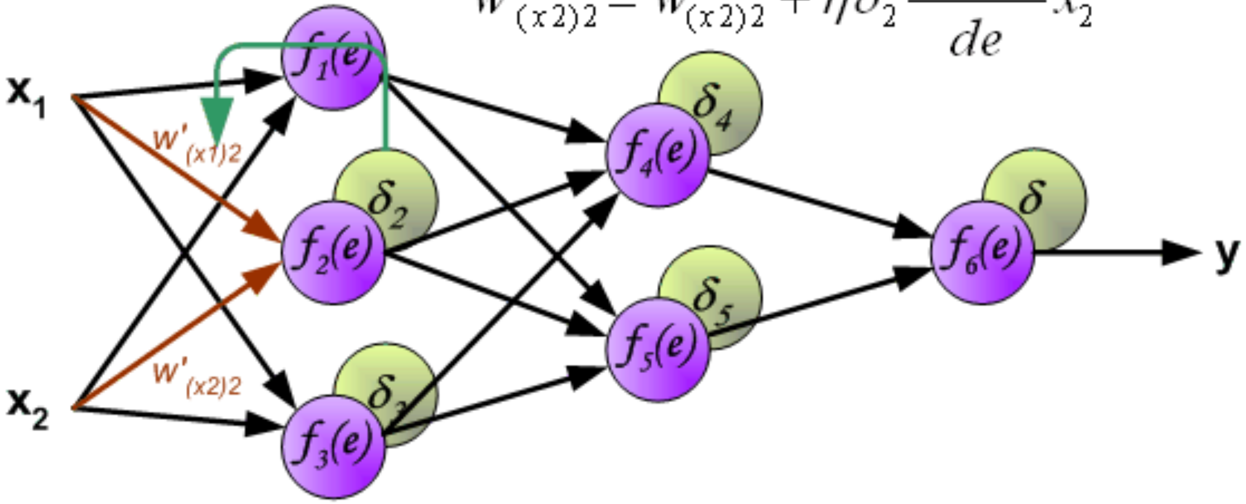
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



1st Layer Refresh

$$w'_{(x1)2} = w_{(x1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2$$



2nd Layer Refresh

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$

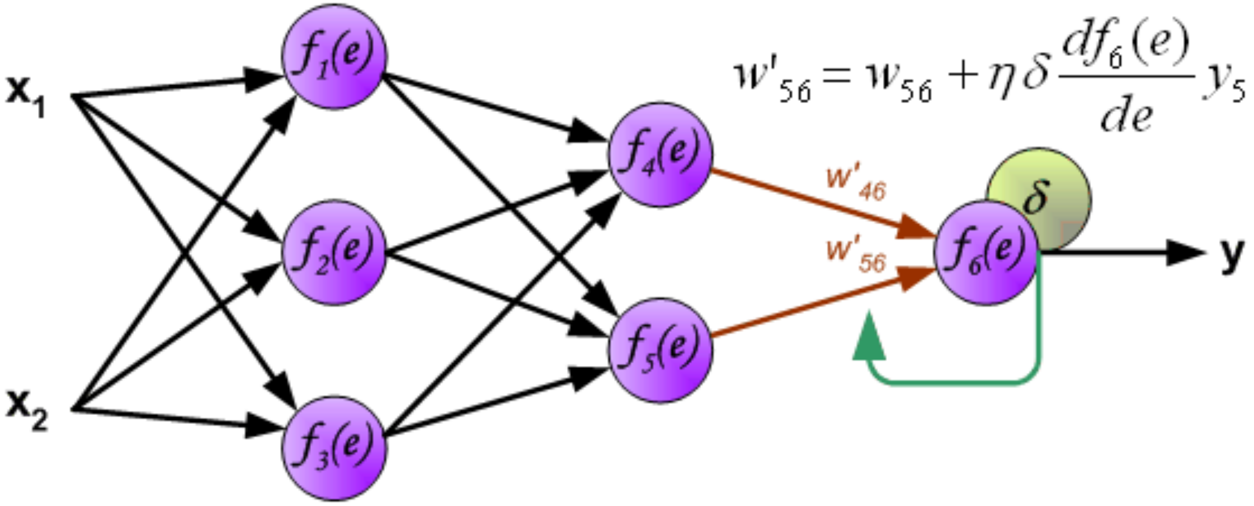
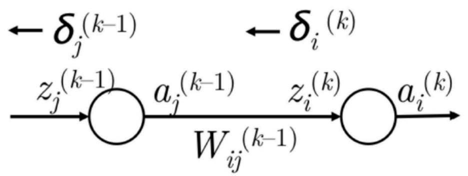
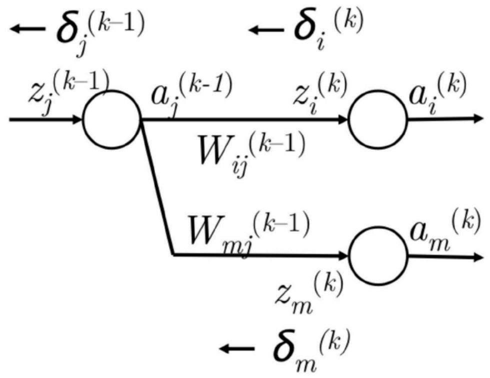


Table of contents

Backpropagation from δ^k to δ^{k-1}





We have error $\delta_i^{(k)}$ propagating backwards from $z_j^{(k)}$, i.e. neuron i at layer k.

We propagate this error backwards to $a_j^{(k-1)}$ by multiplying $\delta_i^{(k)}$ by the path weight $W_{ij}^{(k-1)}$.

Thus, the error received at $a_j^{(k-1)}$ is $\delta_i^{(k)} W_{ij}^{(k-1)}$.

However, $a_j^{(k-1)}$ may have been forwarded to multiple nodes in the next layer(i.e. node m in layer k).

Thus, the total error received at $a_j^{(k-1)}$ is $\delta_i^{(k)} W_{ij}^{(k-1)} + \delta_m^{(k)} W_{mj}^{(k-1)}$.

In fact, we can generalize this to be $\sum_i \delta_i^{(k)} W_{ij}^{(k-1)}$.

Now, we can have the correct error at $a_j^{(k-1)}$, we move it across neuron j at layer $k-1$ by multiplying with the local gradient $\sigma'(z_j^{(k-1)})$.

Finally, the error that reaches at $z_j^{(k-1)}$, called $\delta_j^{(k-1)}$ is $\sigma'(z_j^{(k-1)}) \sum_i \delta_i^{(k)} W_{ij}^{(k-1)}$.

Thanks

FAQ时间