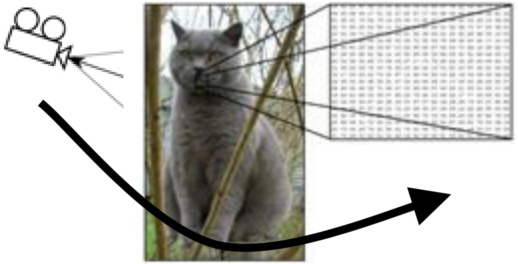
Loss Functions and Optimization. Gradient Descent Algorithm.

Radu Ionescu, Prof. PhD.
raducu.ionescu@gmail.com

Faculty of Mathematics and Computer Science
University of Bucharest

Challenges in Visual Recognition

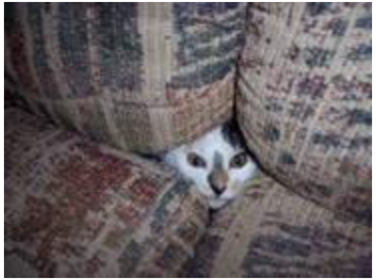
Camera pose



Illumination

Deformation

Occlusion



Background clutter

Intra-class variation

Inter-class similarity

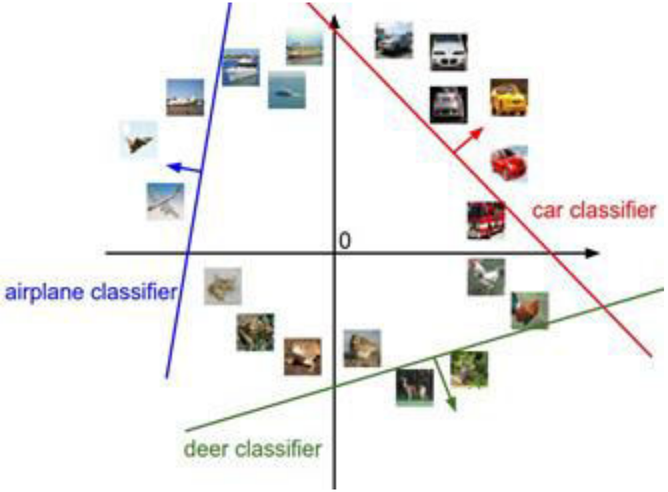
Linear classifier for the multi-class problem

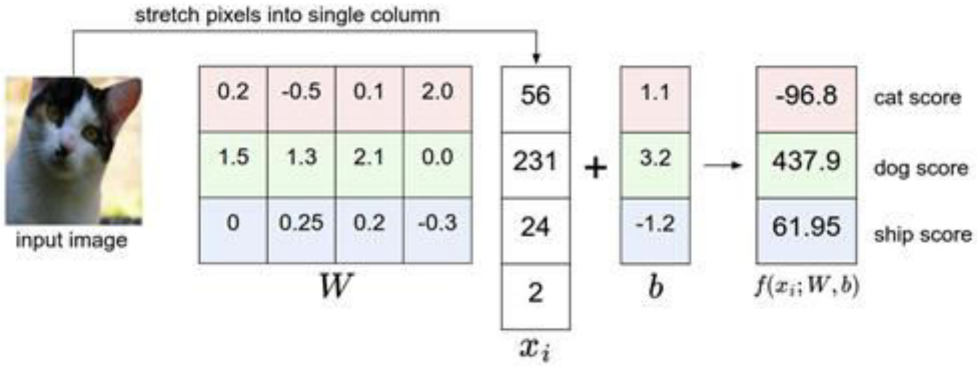
[32x32x3]
array of numbers 0...1
(3072 numbers total)

image parameters

f(**x**,**W**)

10 numbers, indicating class scores





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**

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:

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

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

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	

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, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

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

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:

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

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 10.9)/3
= **4.6**

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$\boxed{s = f(x_i; W)}$$

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes

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

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

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:

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

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

cat	**3.2**
car	5.1
frog	-1.7

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

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

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

Softmax Classifier (Multinomial Logistic Regression)

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

cat	**3.2**
car	5.1
frog	-1.7

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

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

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

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

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

cat	**3.2**		**24.5**		**0.13**	→ $L_i = -\log(0.13)$ $=$ **0.89**
car	5.1	$\xrightarrow{\text{exp}}$	164.0	$\xrightarrow{\text{normalize}}$	0.87	
frog	-1.7		0.18		0.00	

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)

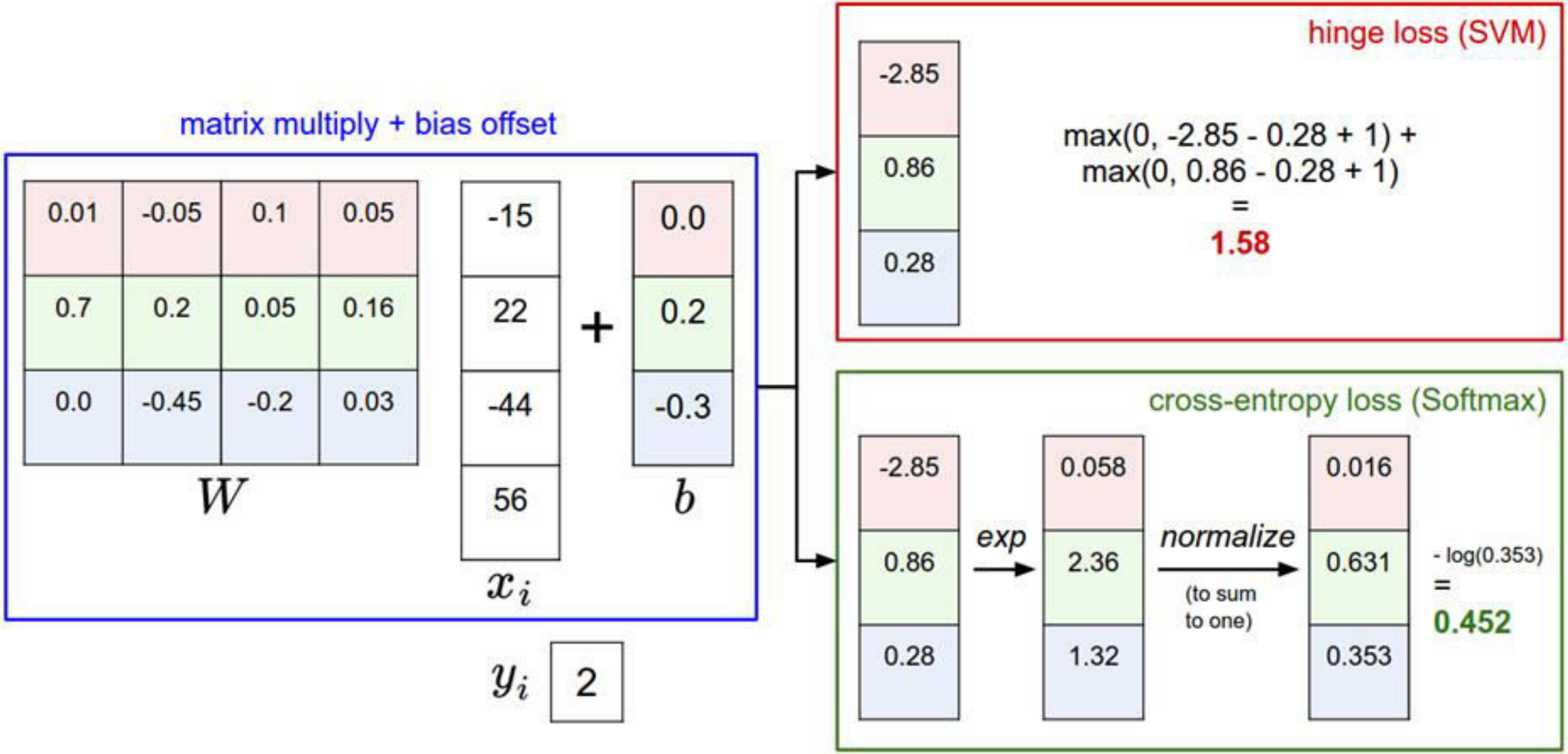
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q: What is the min/max possible loss L_i?

	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

$\rightarrow L_i = -\log(0.13)$
$= \mathbf{0.89}$


matrix multiply + bias offset
0.01
-0.05
0.1
0.05
0.7
0.2
0.05
0.16
0.0
-0.45
-0.2
0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
y_i
2
hinge loss (SVM)
-2.85
0.86
0.28
max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum to one)
0.016
0.631
0.353
- log(0.353)
=
0.452

Softmax vs. SVM

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

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

Softmax vs. SVM

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

assume scores:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and $\boxed{y_i = 0}$

Q: Suppose we take a datapoint and we add some small perturbations (changing its score slightly). What happens to the loss in both cases?

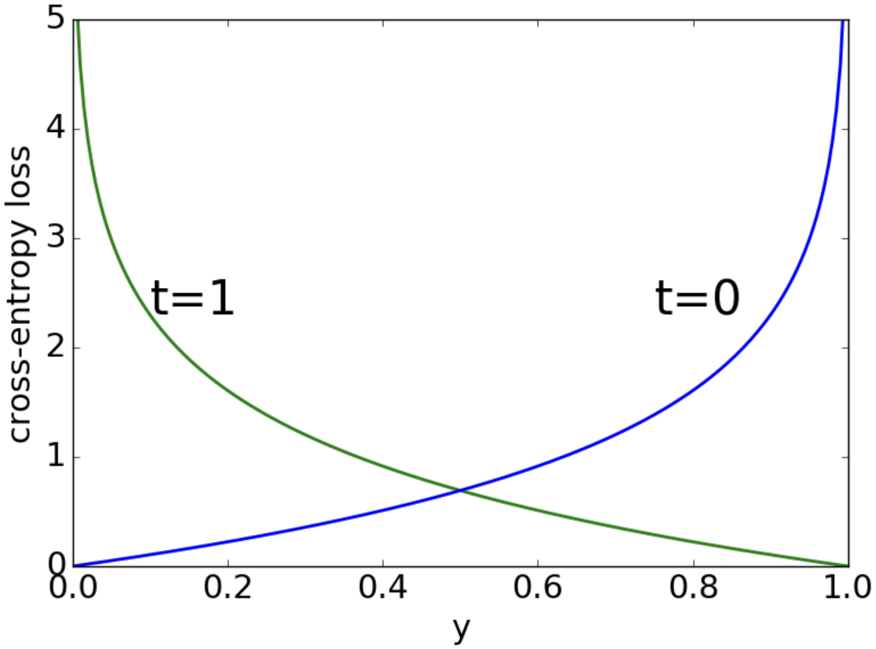
Binary cross-entropy loss

- Binary cross-entropy (logistic) loss:

$$L_i = -(t_i \cdot log(y_i) + (1 - t_i) \cdot log(1 - y_i))$$

where t_i is the ground-truth binary label (0 or 1) of sample x_i,

and y_i is the prediction for the same sample



Optimization

Until now:

- 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**:

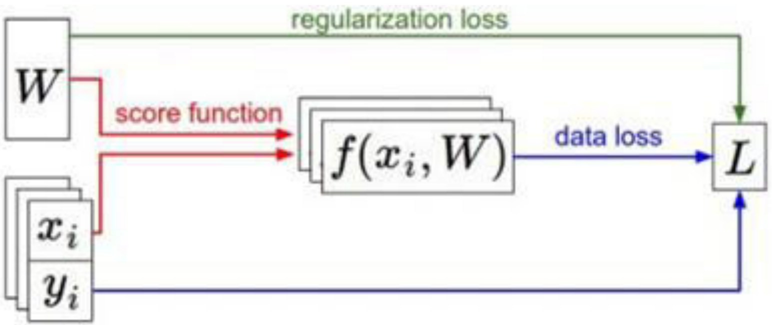
Softmax

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

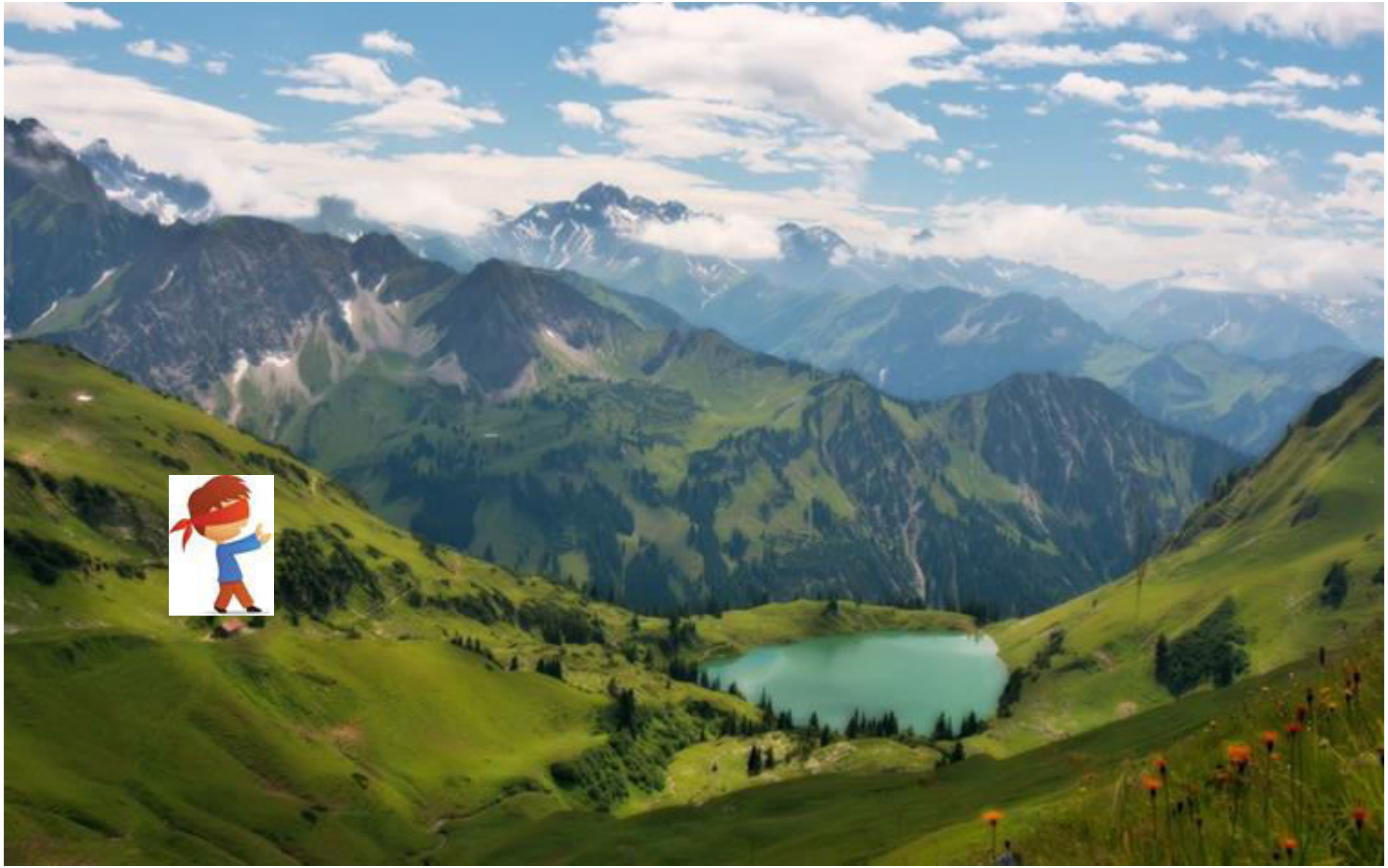
SVM

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

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



Gradient Descent Algorithm



Gradient Descent Algorithm

- 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.

current W:

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

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

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:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

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

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

?,
?,…]

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	?,
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 1.25353**	

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 (second dim):

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

loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,
?,…]

(1.25353 - 1.25347)/0.0001
= 0.6

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

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
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 1.25347**	

current W:	W + h (third dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	**0**,
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 1.25347**	

(1.25347 - 1.25347)/0.0001
= 0

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

Gradient Evaluation

1) Numerical approach

We choose a small positive h and apply the formula:

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

- We obtain an approximate value
- Very slow to compute

2) Analytic approach

We use calculus to determine the gradient’s formula as a function of X and W

Gradient Evaluation (Python)

```
def f(x):
    y = 0.5 * (x**4) - 2 * (x**2) + x + 5
    return y
# 1) Numerical Method
h = 0.001
gradient = (f(x + h) - f(x)) / h
# 2) Analythic Method
def f_prime(x):
    y_prime = 2 * (x**3) - 4 * x + 1
    return y_prime
gradient = f_prime(x)
```

current W:

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

dW = ...
(some function of x
and W)

gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

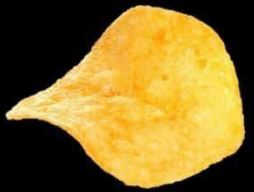
In summary:

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone

=>

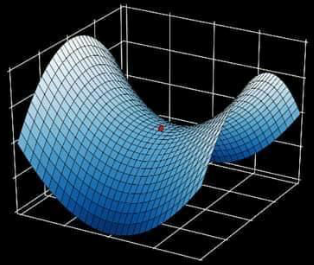
In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient checking.**

What others see



Potato chip

What i see



Gradient descent

Gradient descent is an iterative optimization algorithm for finding the **minimum** of a function.

next position — *opposite direction* — *gradient at current position*

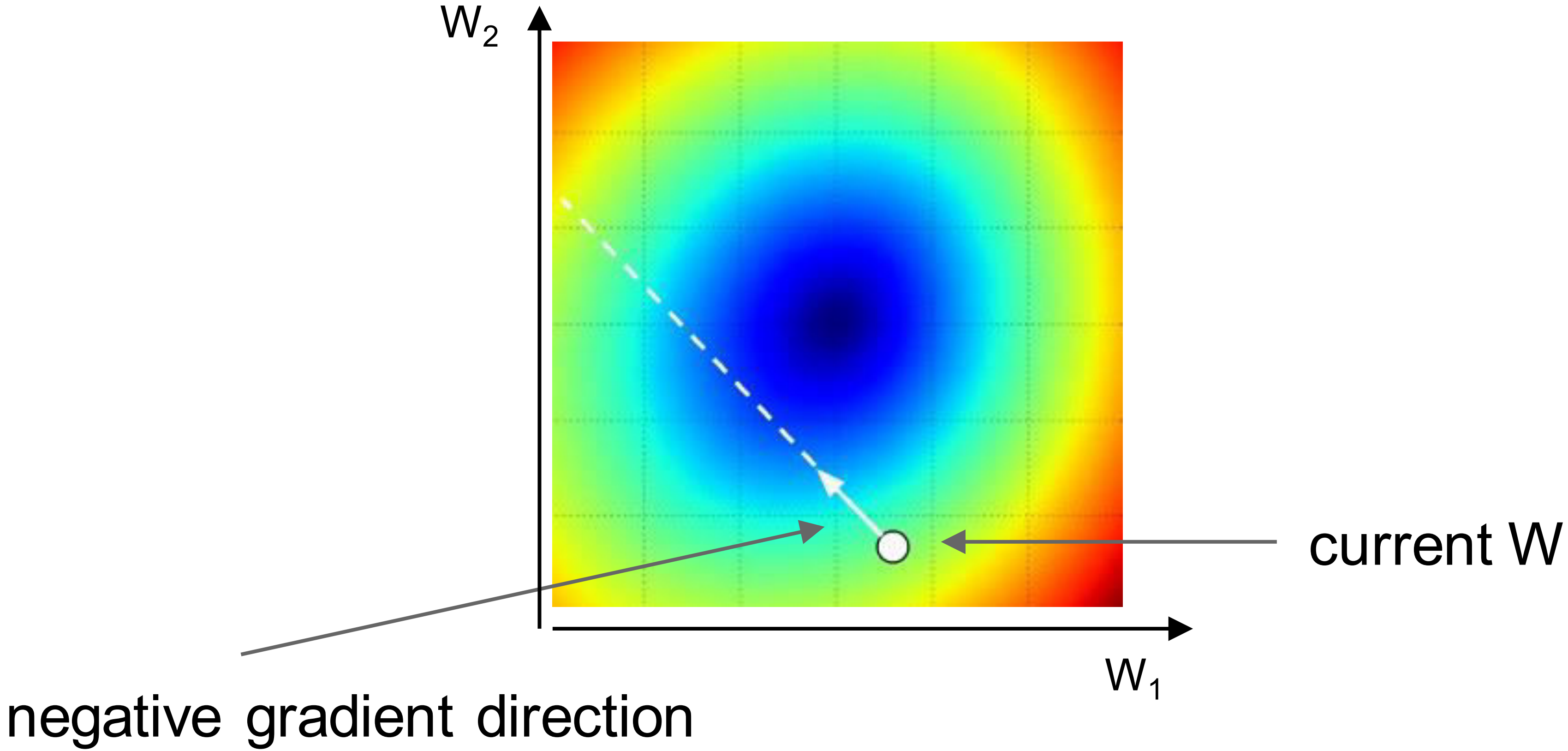
$$x_{i+1} = x_i - \alpha \nabla f(x_i)$$

current position — *step size (learning rate)*

Gradient Descent (Python)

```
def GD(W0, X, goal, learningRate):
  perfGoalNotMet = true
  W = W0

  while perfGoalNotMet:
    gradient = eval_gradient(X, W)
    W_old = W
    W = W – learningRate * gradient
    perfGoalNotMet = sum(abs(W - W_old)) > goal
```



W2
W1
current W
negative gradient direction

Mini-batch Gradient Descent

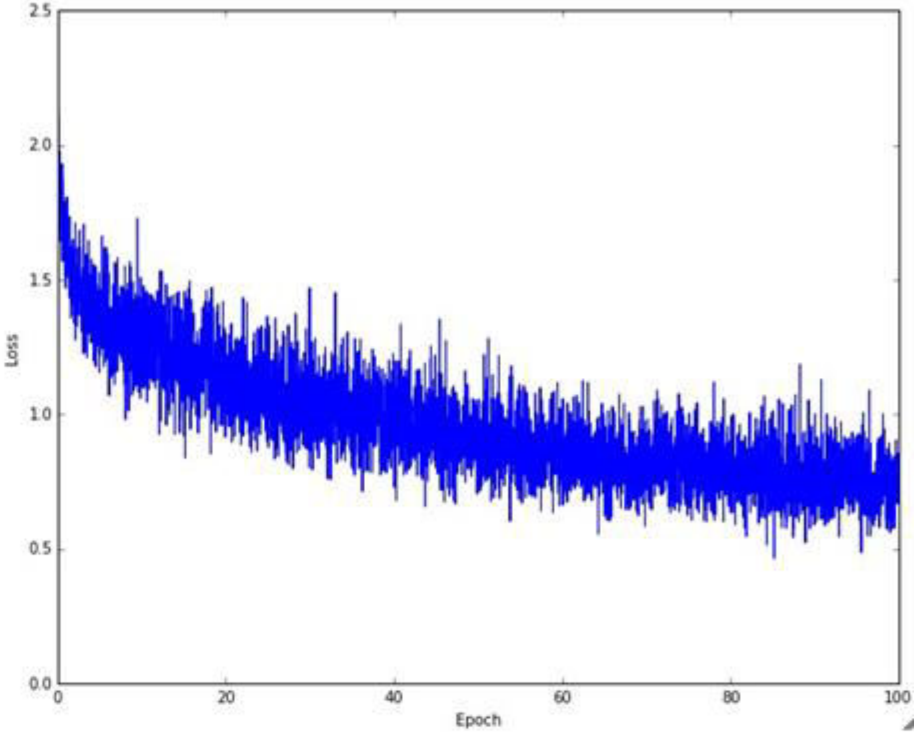
- Also known as **Stochastic Gradient Descent (SGD)**
- Only use a small portion of the training set to compute the gradient:

```
. . .
while perfGoalNotMet:

        X_batch = select_random_subsample(X)
        gradient = eval_gradient(@loss, X_batch, W)
        . . .
```

- Common mini-batch sizes are 32/64/128 examples

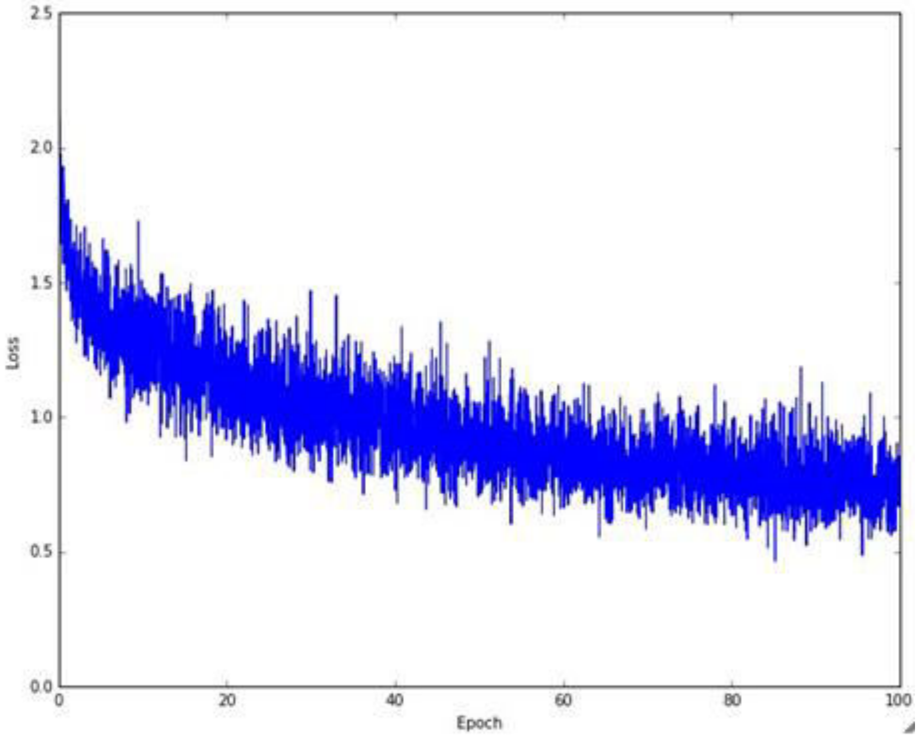
e.g. Krizhevsky’s ILSVRC ConvNet used 256 examples

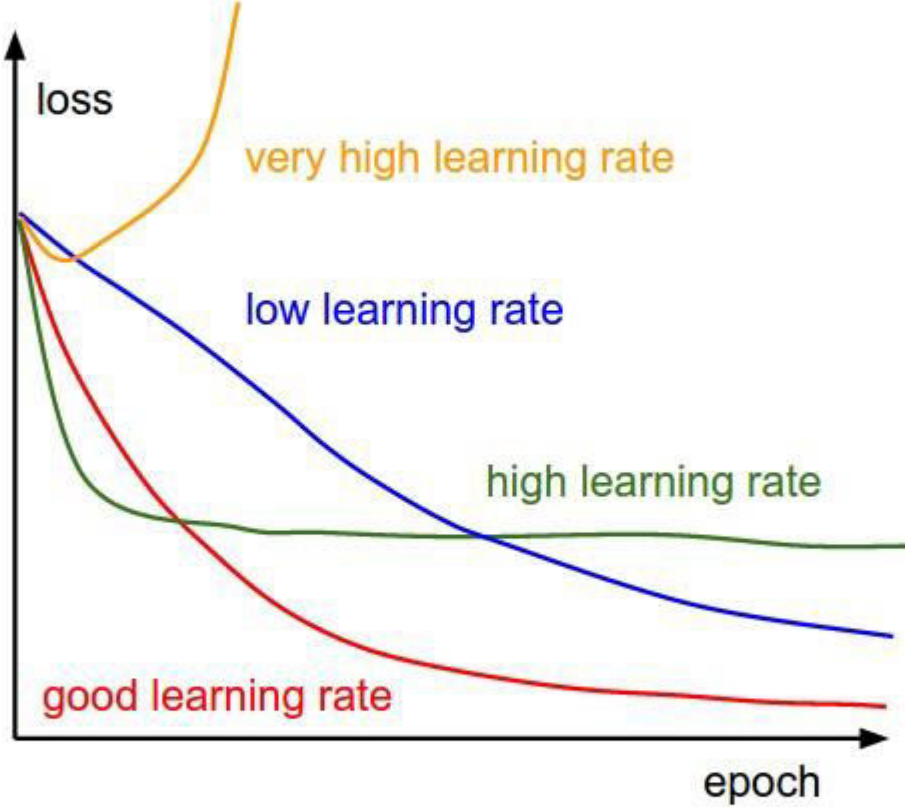


Example of optimization progress while training a neural network with SGD.

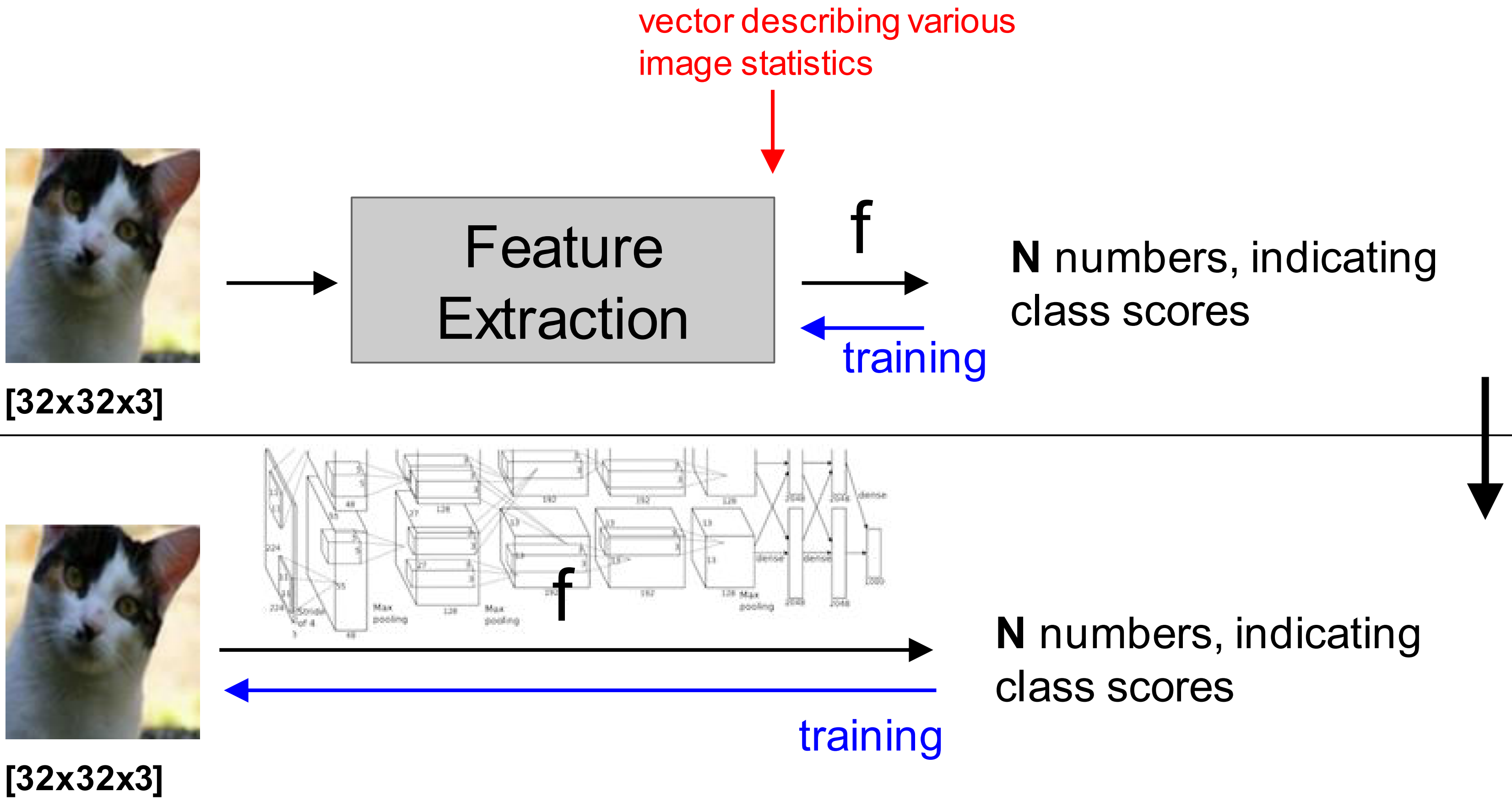
(Loss over mini-batches goes down over time)

The effects of step size (or “learning rate”)

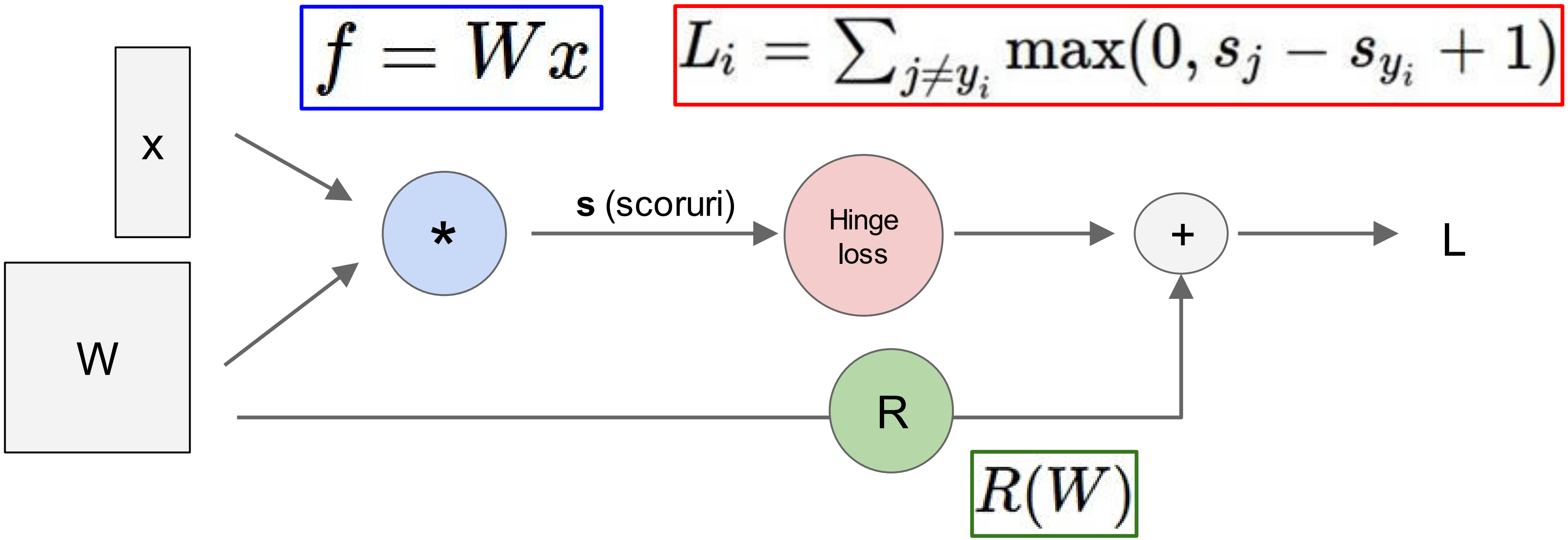




From feature extract to end-to-end learning



Computational Graph

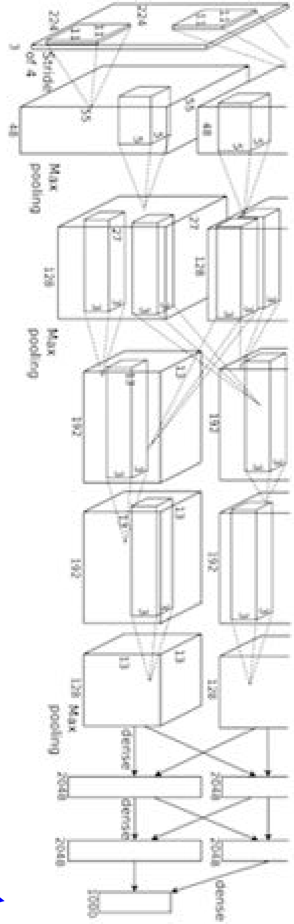


Convolutional Network (AlexNet)

input image

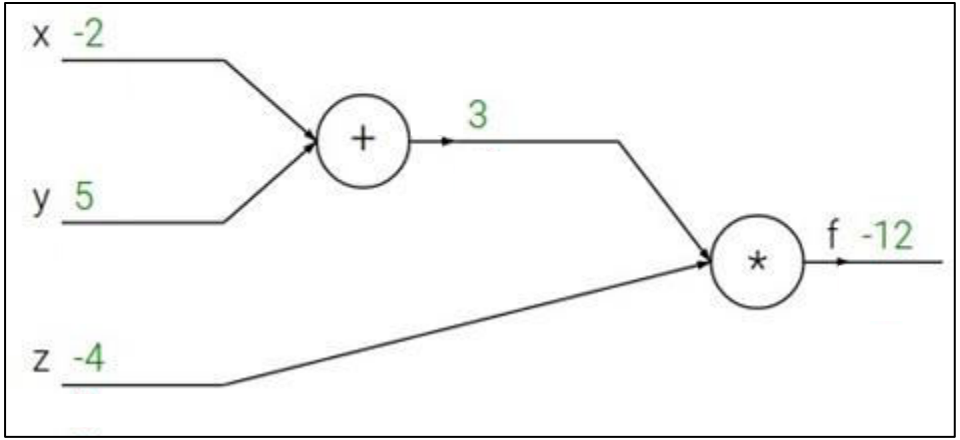
weights

loss



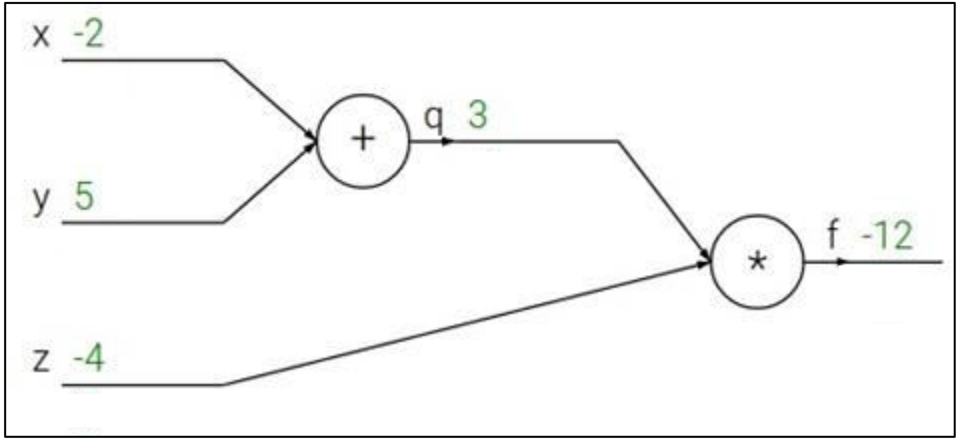
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

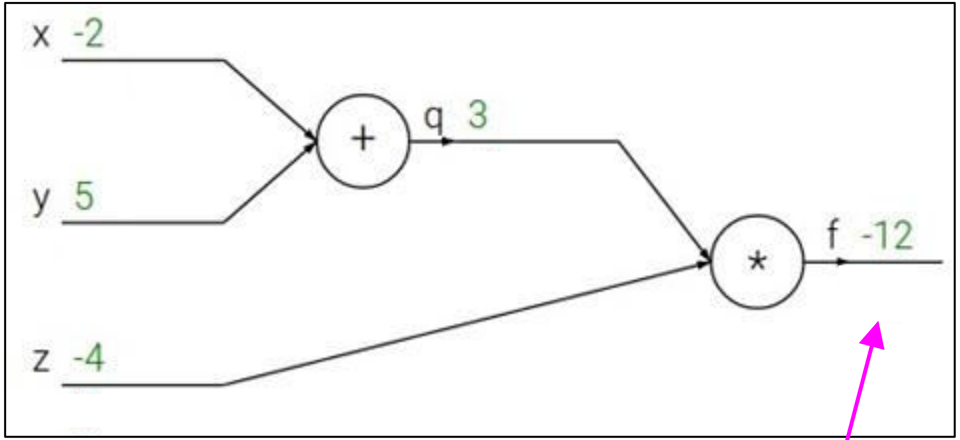
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\boxed{\frac{\partial f}{\partial f}}$

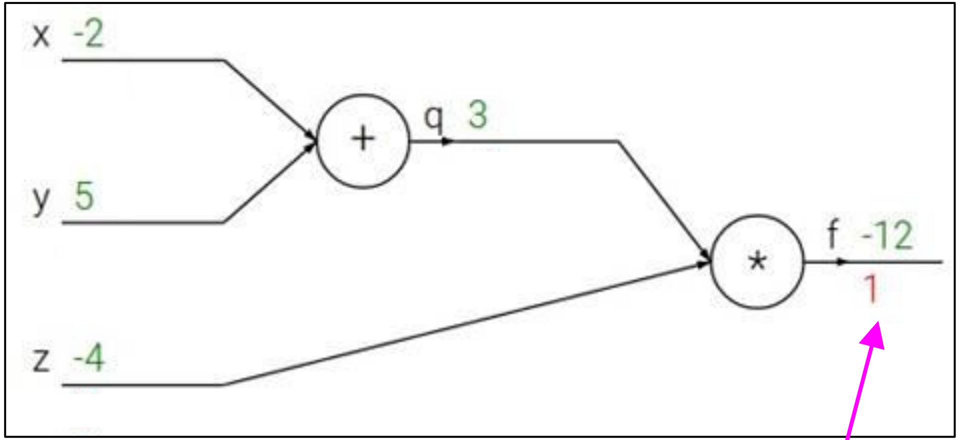
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

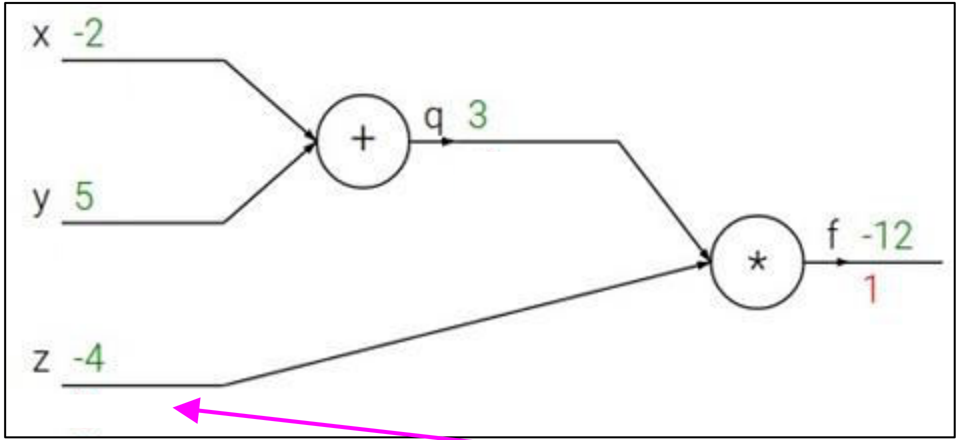
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial f}$$

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial z}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

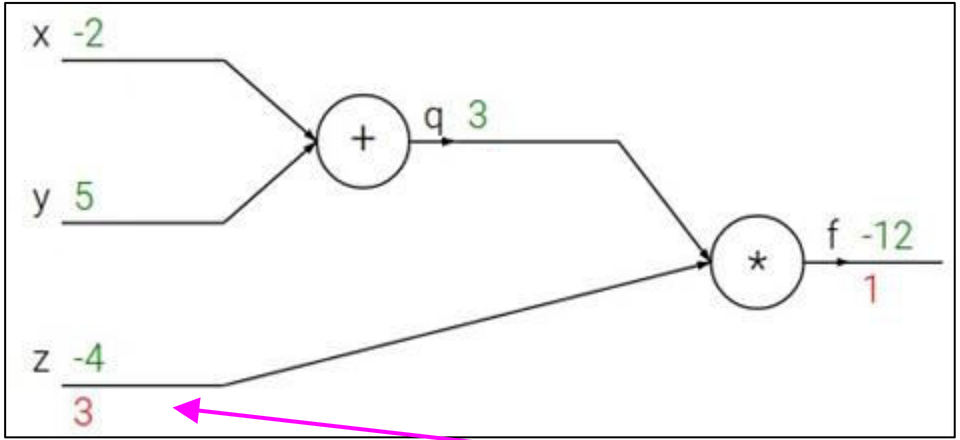
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

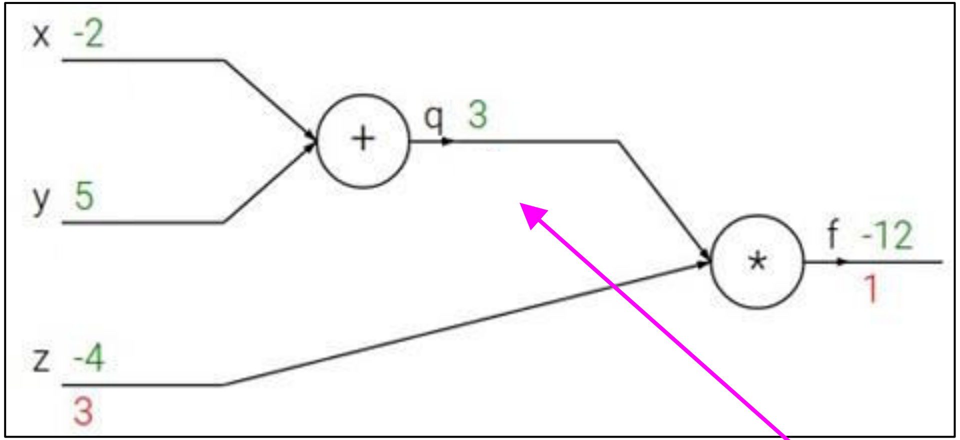
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial q}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

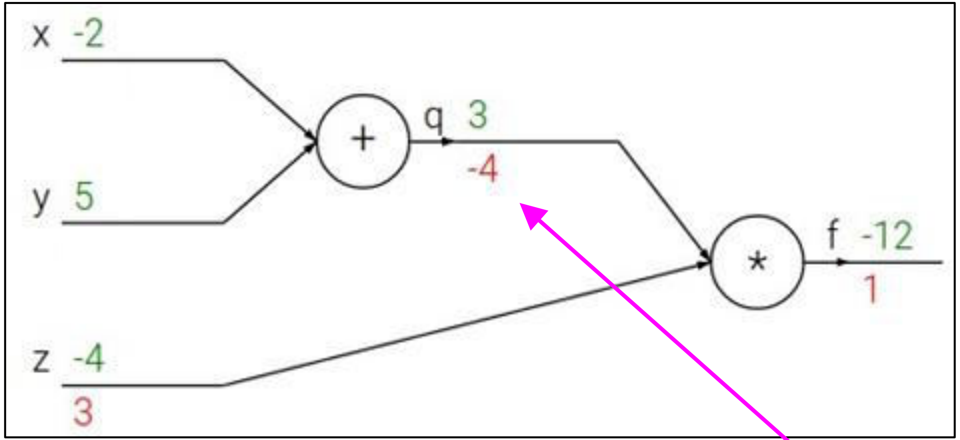
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

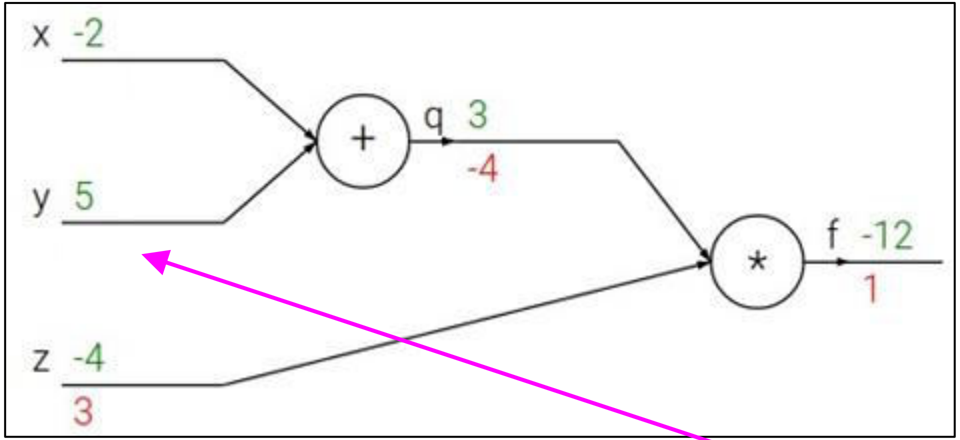
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial q}}$$

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial y}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

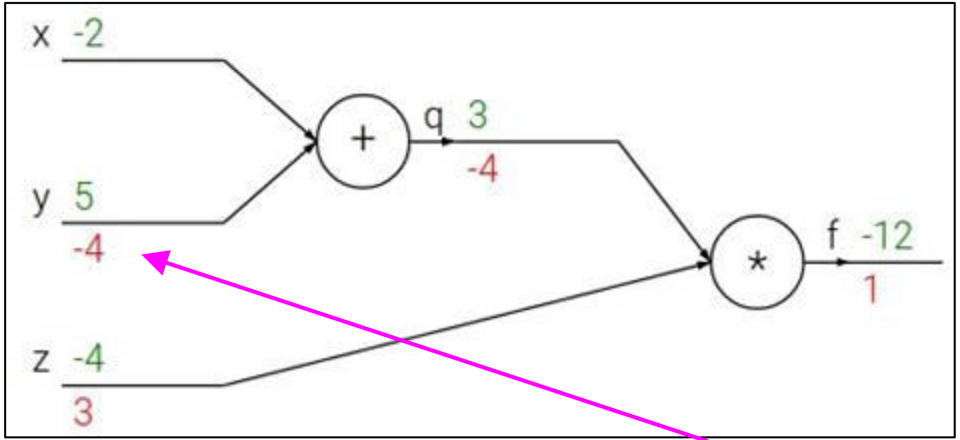
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial y}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}}$$

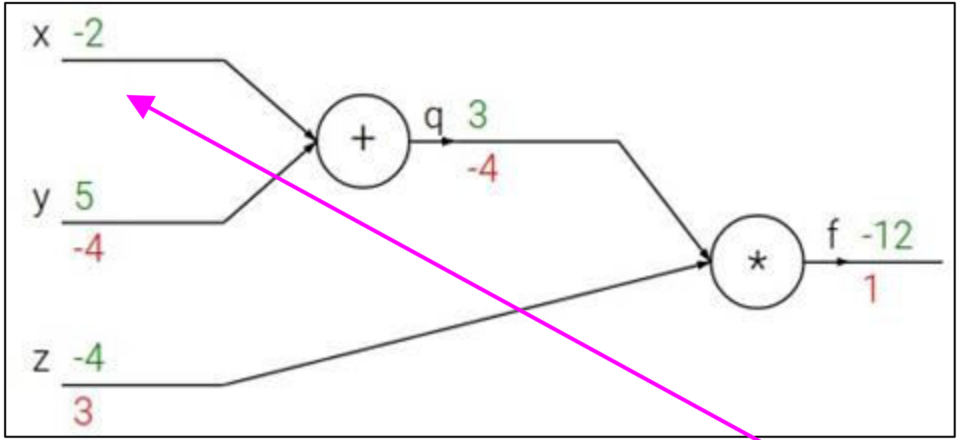
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

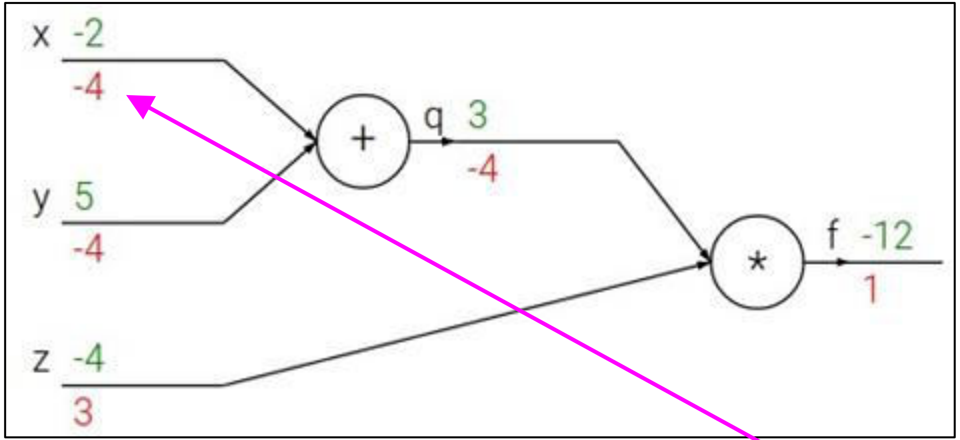
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

vrem: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

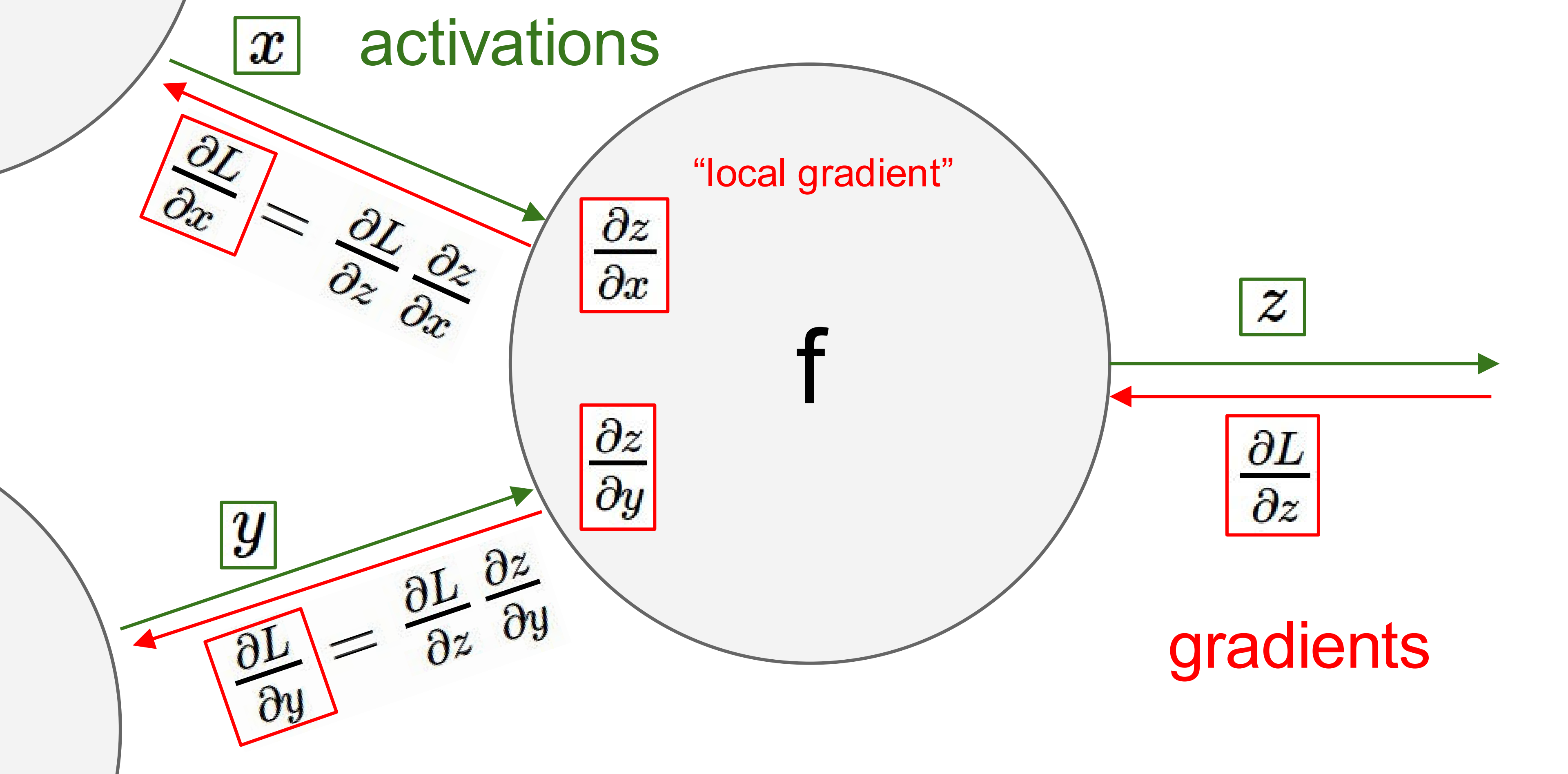


$\frac{\partial f}{\partial x}$

Chain rule:

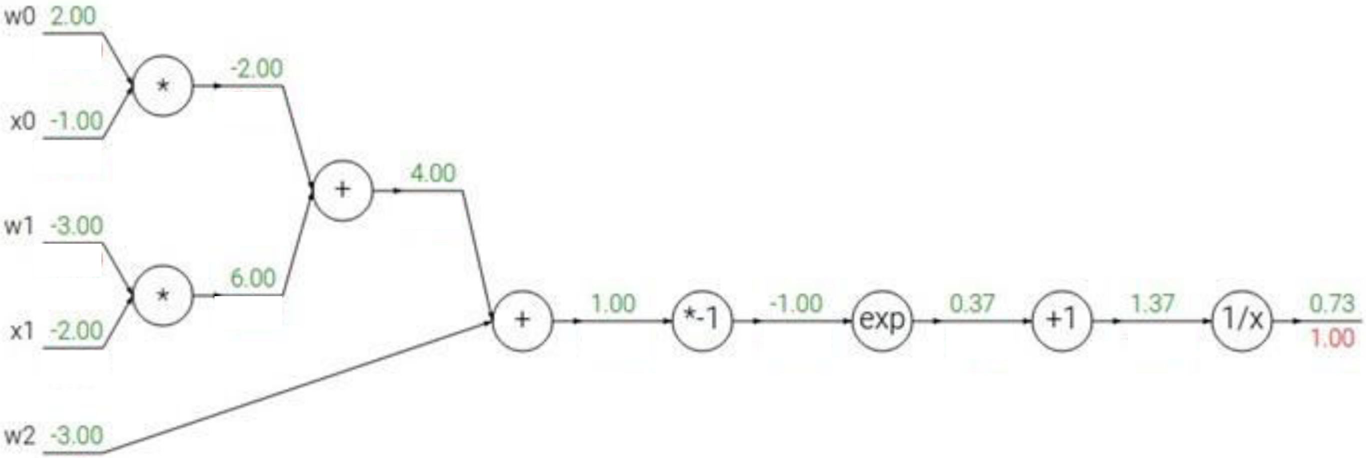
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}}$$

Gradient propagation using chain rule



Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

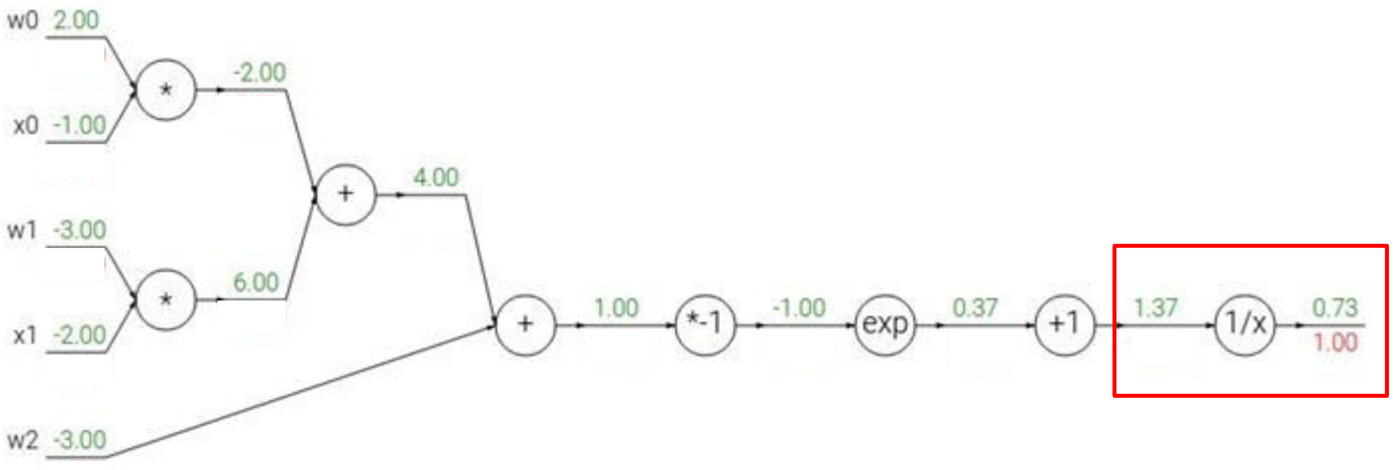
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

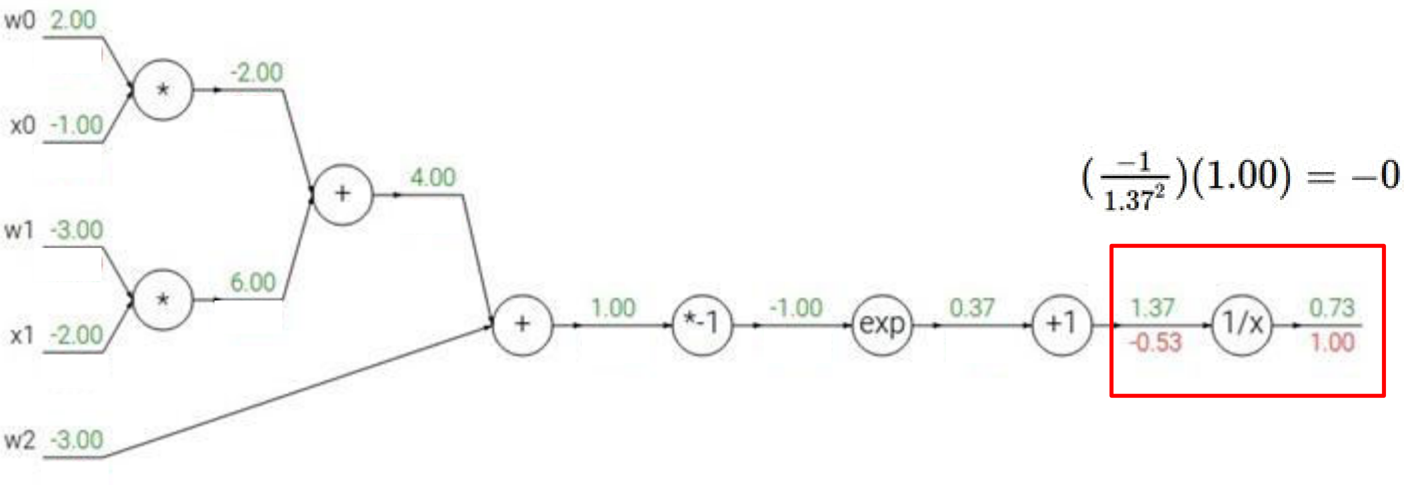
$$\boxed{f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\left(\frac{-1}{1.37^2}\right)(1.00) = -0.53$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

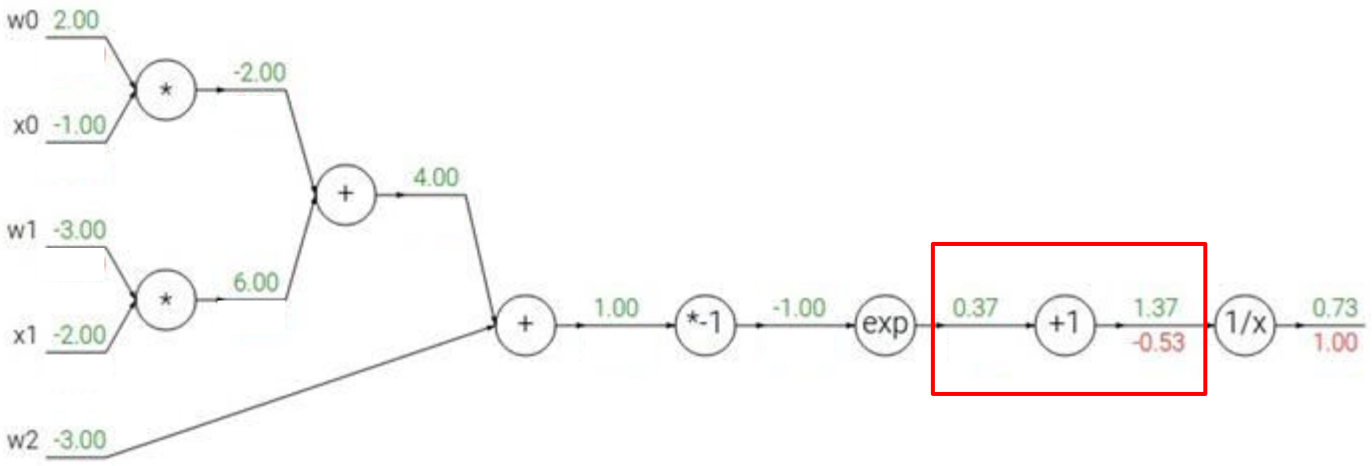
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$\boxed{f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

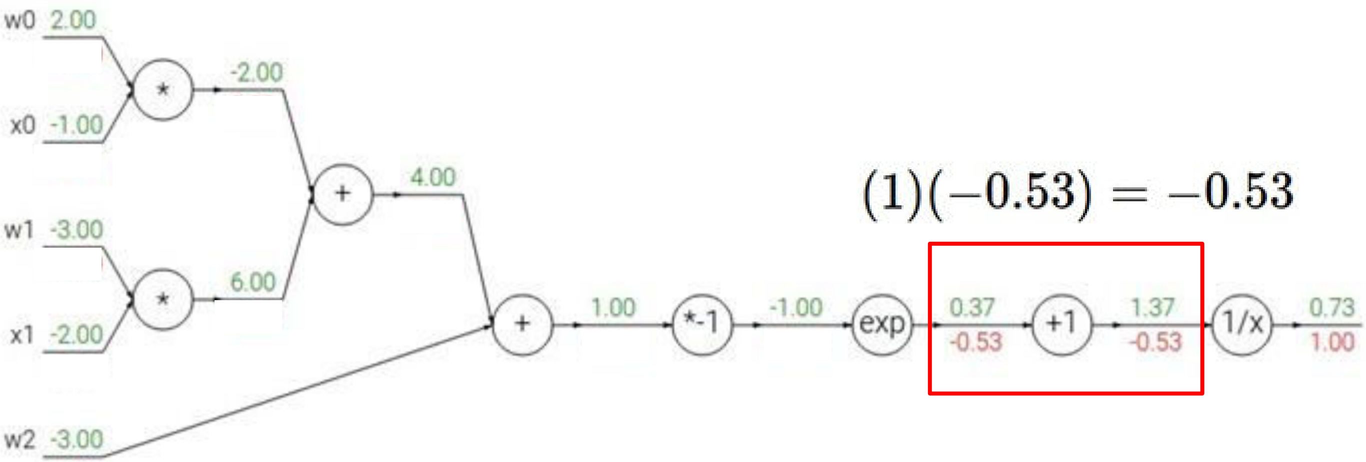
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$\boxed{f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1}$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(1)(-0.53) = -0.53$$

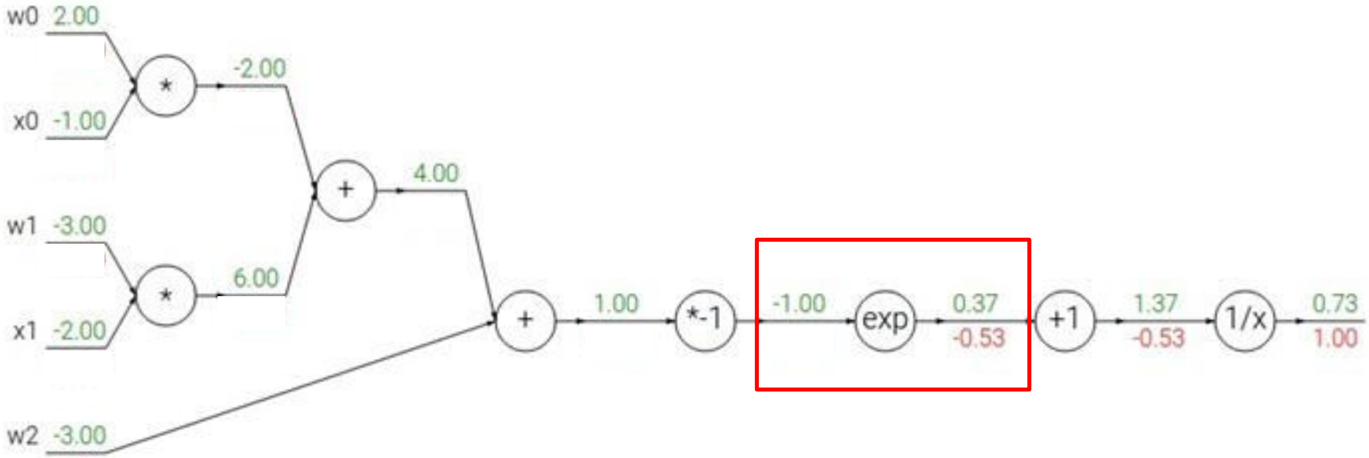
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$\boxed{f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1}$$

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$$\boxed{f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x}$$

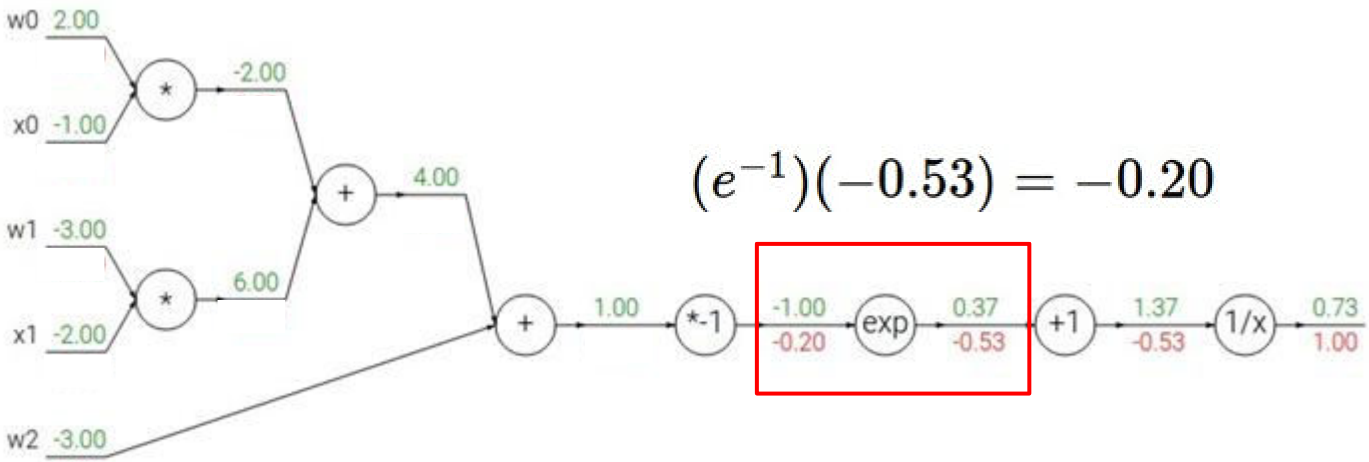
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(e^{-1})(-0.53) = -0.20$$

$$\boxed{f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x}$$

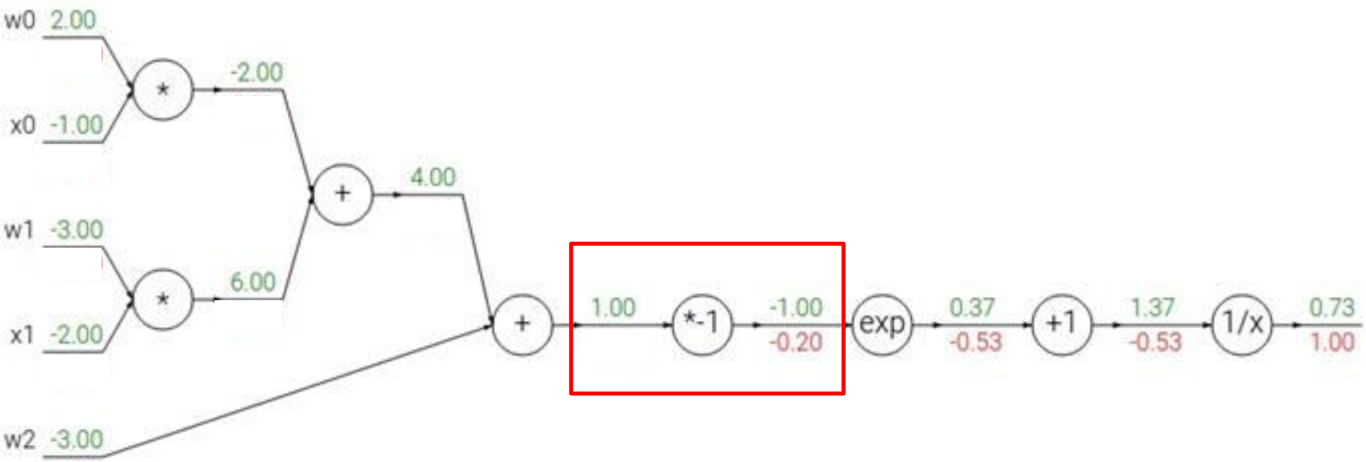
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

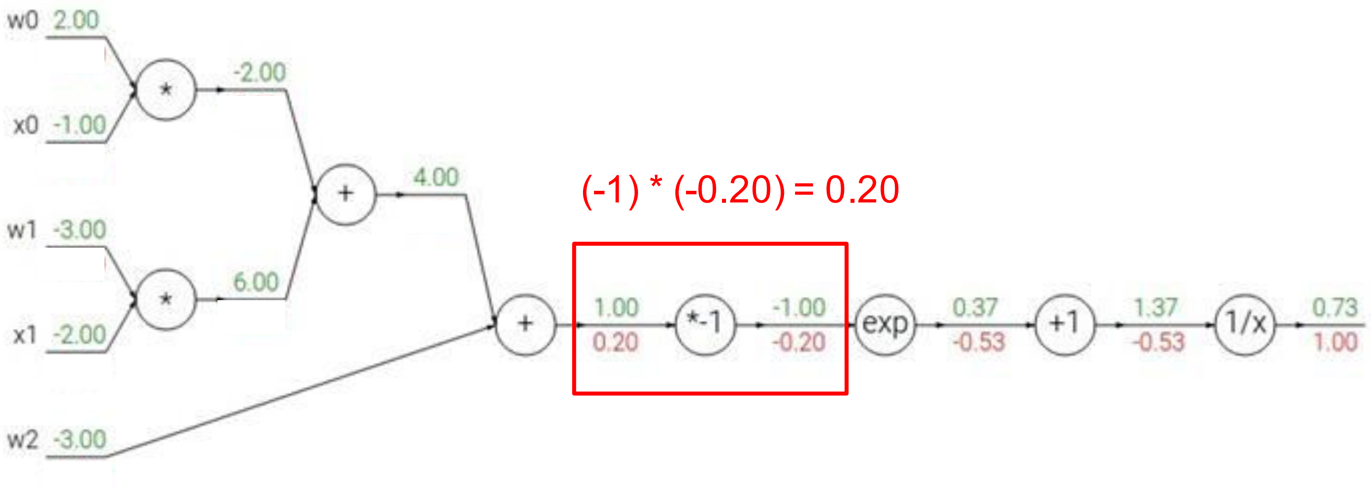
$$\boxed{f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a}$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

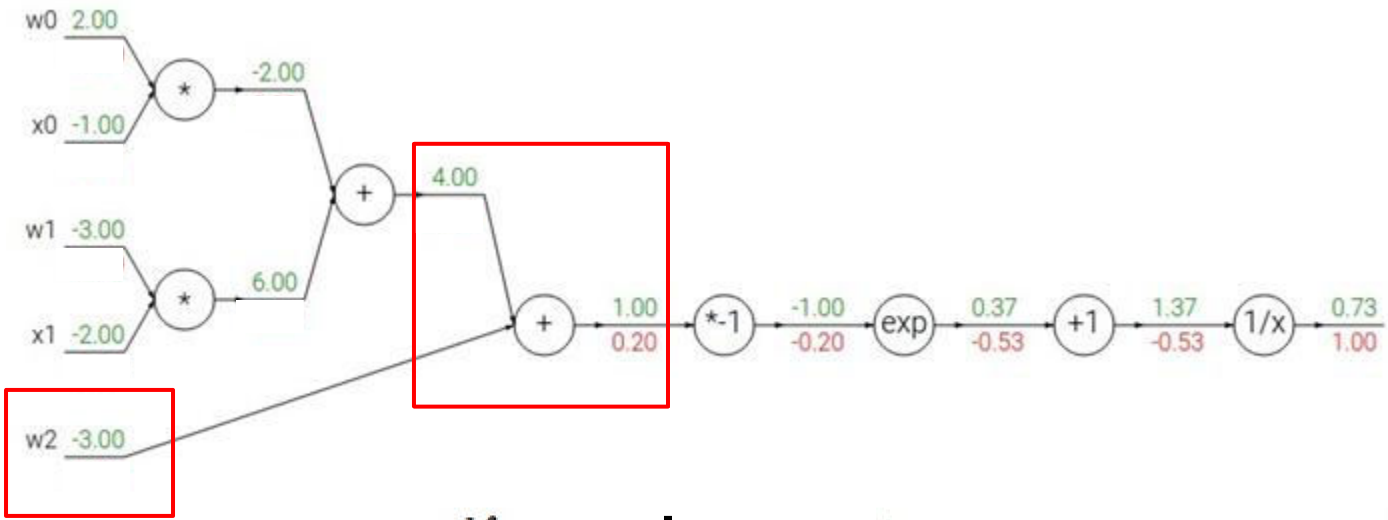
$$\boxed{f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a}$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

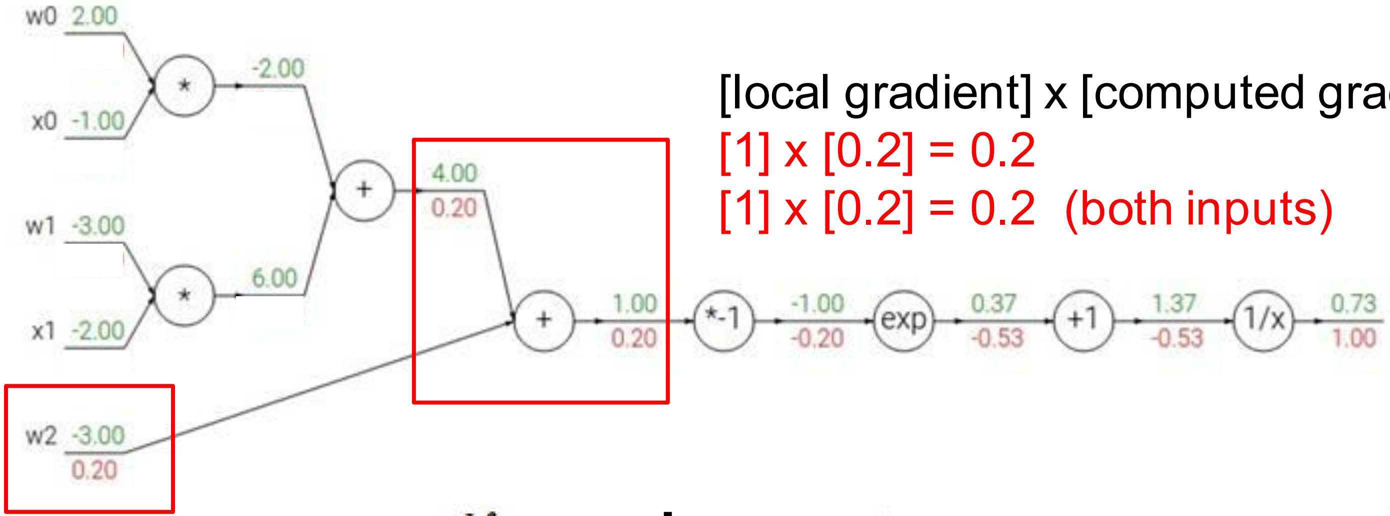
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [computed gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs)

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

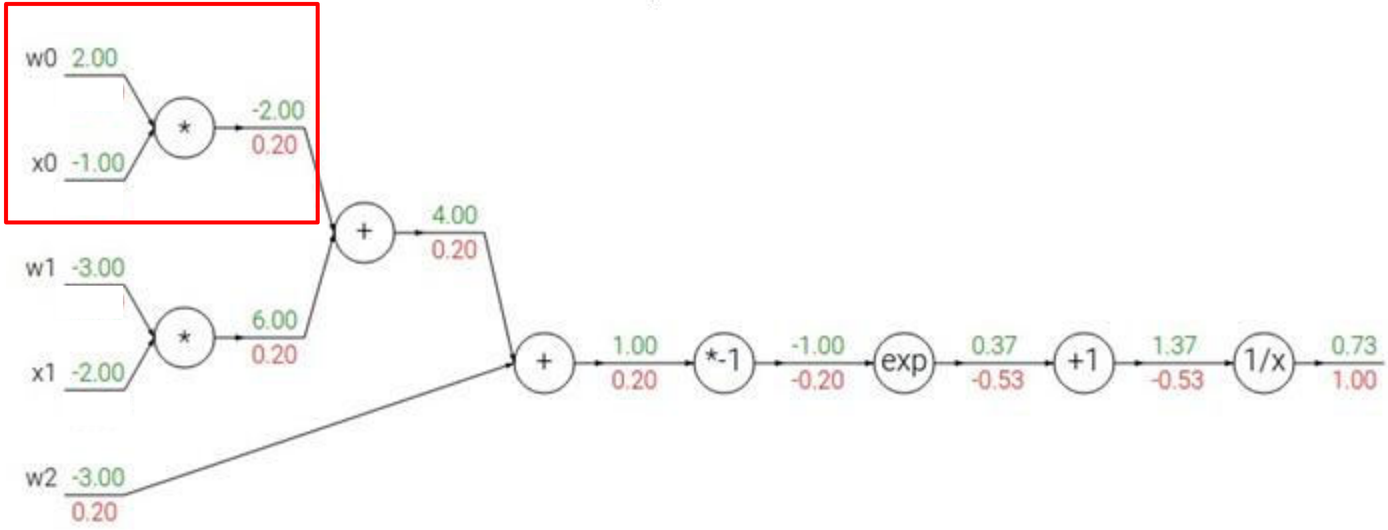
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Un alt exemplu:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

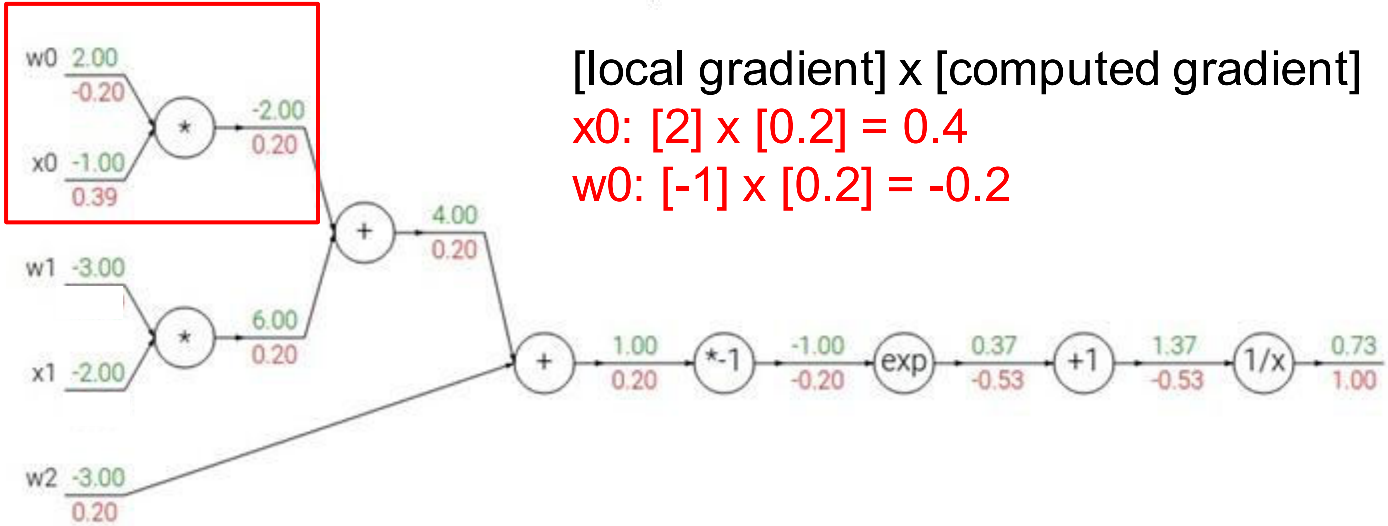
Un alt exemplu:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

[local gradient] x [computed gradient]

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

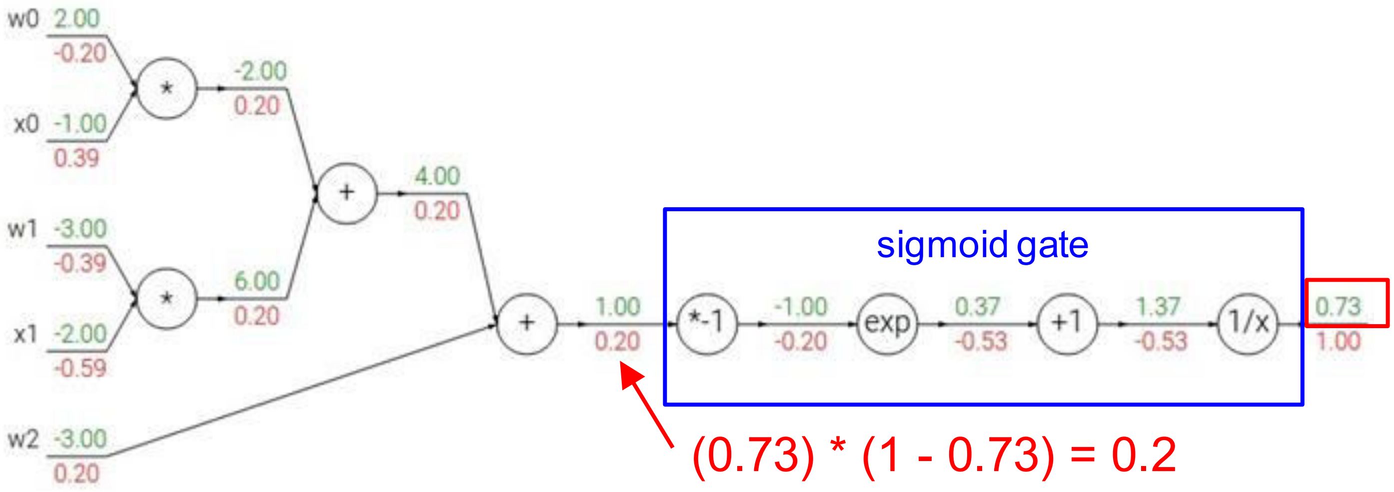
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

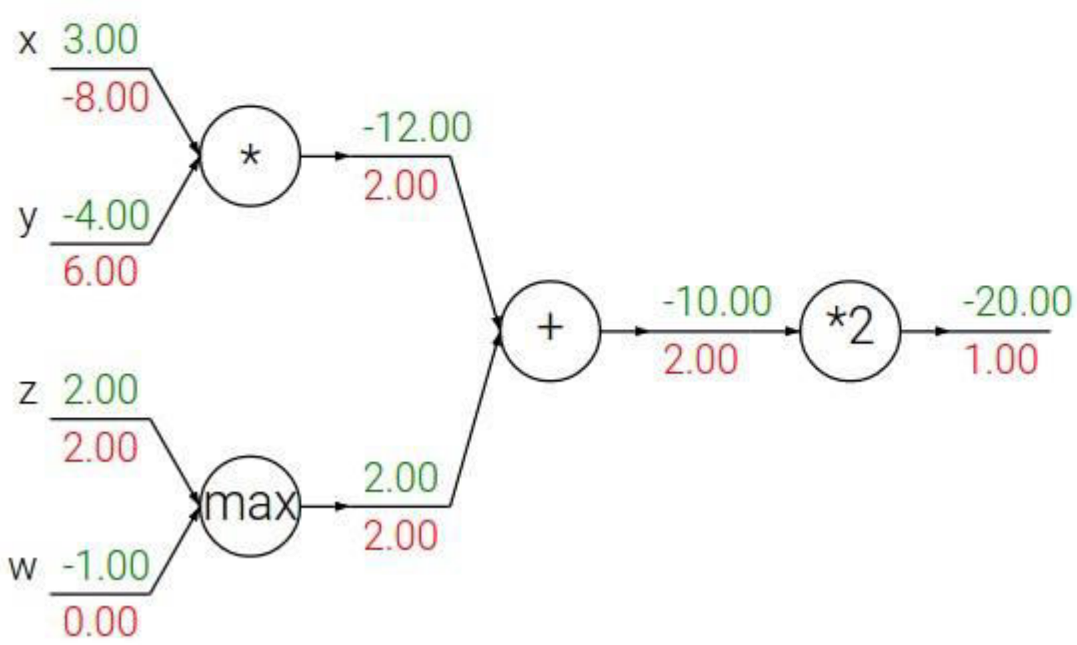


Patterns in backward prop

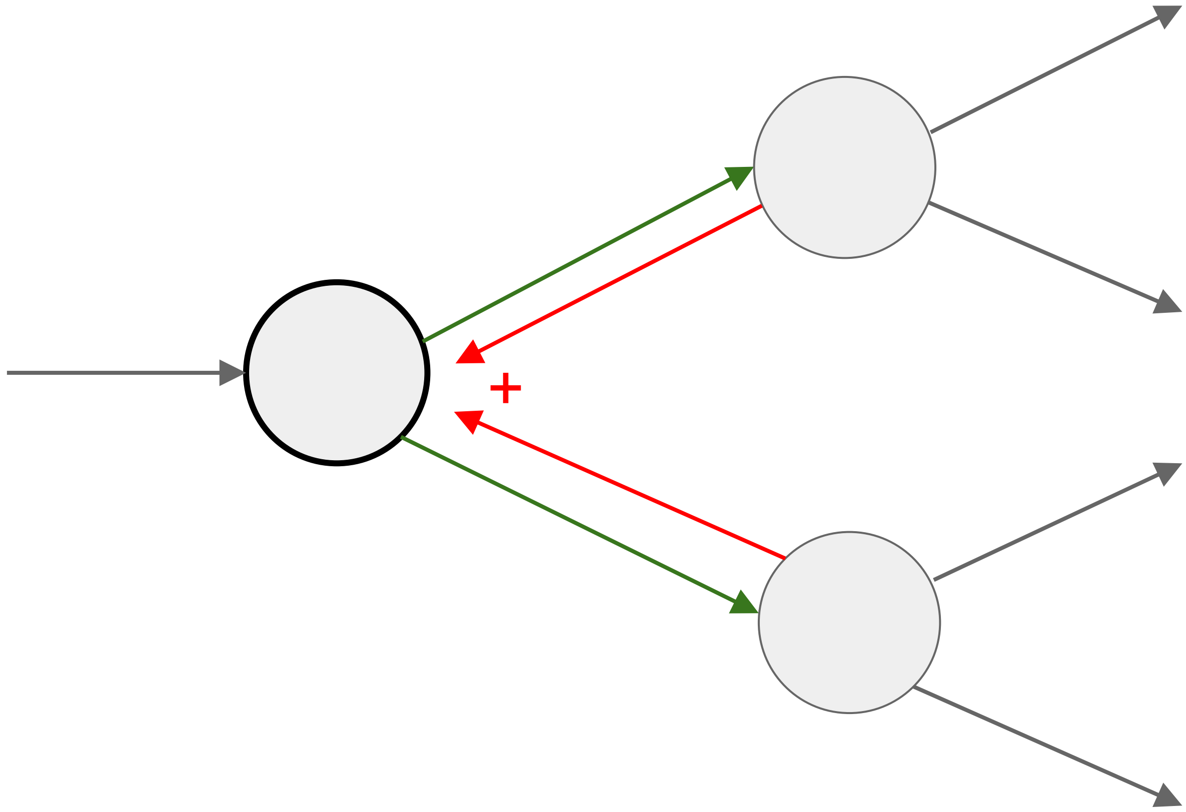
add gate: gradient distributor

max gate: gradient router

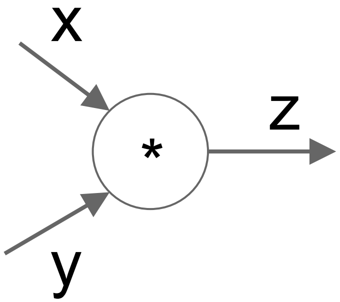
mul gate: gradient… "switcher"?



Gradients add at branches



Forward/backward propagation for **mul** gate (Python)



(x, y, z are scalars)

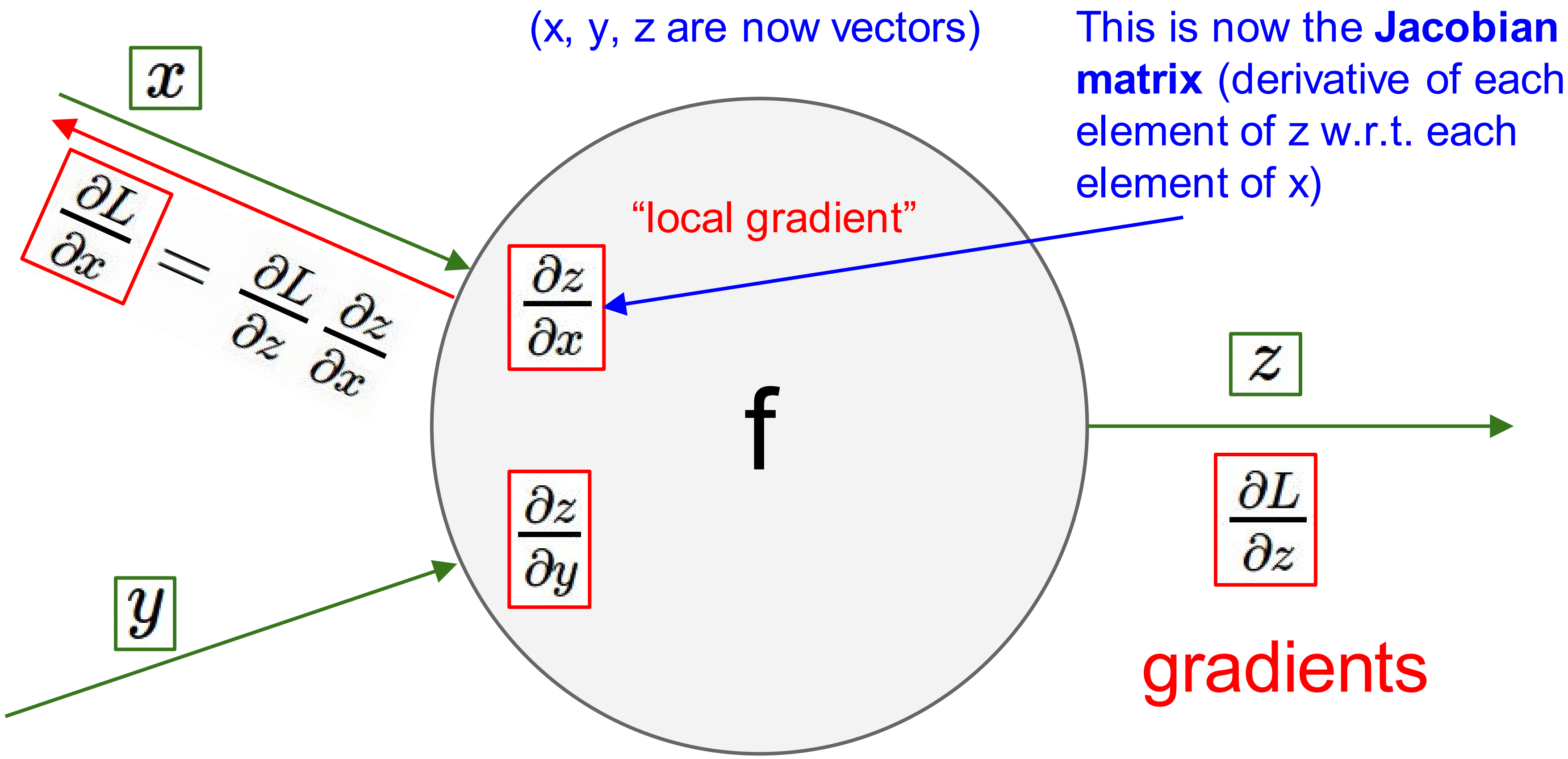
```python
def forward(x,y):
    z = x * y
    layer.input = [x, y] # for backward
    return z
```

$$\boxed{\frac{\partial L}{\partial z}}$$

```python
def backward(dz):
    dx = layer.input[1] * dz # dz/dx * dL/dz
    dy = layer.input[0] * dz # dz/dy * dL/dz
    return [dx, dy]
```

$$\boxed{\frac{\partial L}{\partial x}}$$

Gradients for vectorial code

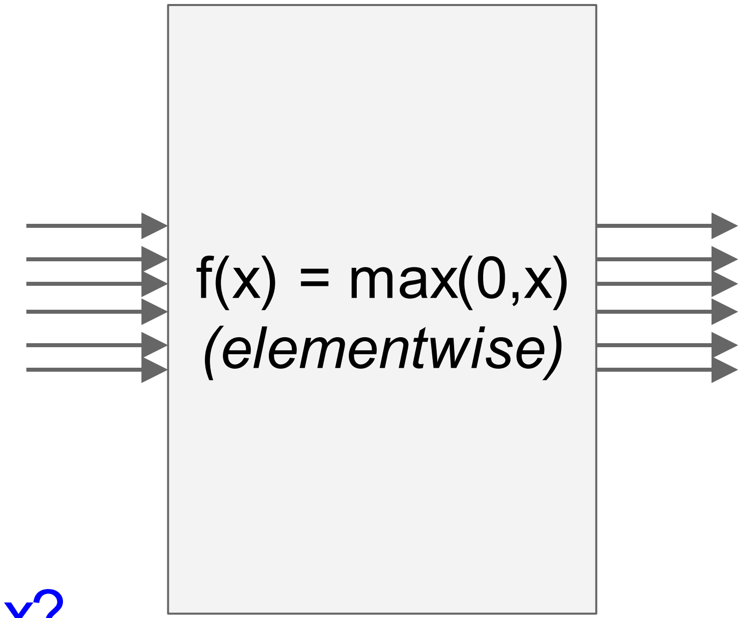


Vectorized operations

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Jacobian matrix

4096-d
input vector



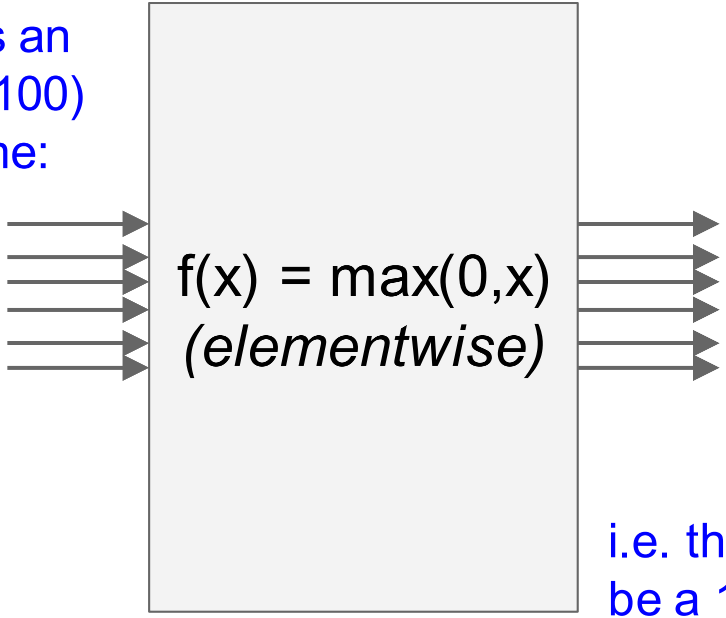
4096-d
output vector

Q: what is the size of the Jacobian matrix?
[4096 x 4096]

Vectorized operations

In practice we process an entire minibatch (e.g. 100) of examples at one time:

100 4096-d input vectors



100 4096-d output vectors

i.e. the Jacobian would technically be a 100 x [4096 x 4096] matrix

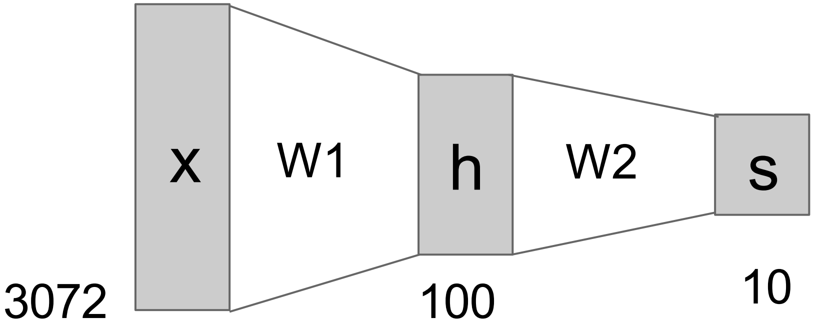
Summary so far

- Neural nets will be very large: no hope of writing down gradient formula by hand for all parameters
- **Backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs / parameters / intermediates
- Implementations maintain a graph structure, where the nodes implement the **forward**() / **backward**() API
- **Forward**: compute result of an operation and save any intermediates needed for gradient computation in memory
- **Backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs

Neural Network: without the brain stuff

(**Before**) Linear score function: $$f = Wx$$

(**Now**) 2-layer Neural Network $$f = W_2 \max(0, W_1 x)$$



Neural Network: without the brain stuff

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Training a 2-layer Neural Network needs ~11 lines (Python)

```
X = np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
Y = np.array([[0,1,1,0]]).T

W0 = 2 * np.random.random((3,4)) - 1
W1 = 2 * np.random.random((4,1)) - 1

for i in range(5000):

    # forward pass
    l1 = 1 / (1 + np.exp(-np.matmul(X, W0)))
    l2 = 1 / (1 + np.exp(-np.matmul(l1, W1)))

    # backward pass
    delta_l2 = (Y - l2) * (l2 * (1 - l2))
    delta_l1 = np.matmul(delta_l2, W1.T) * (l1 * (1 - l1))

    # gradient descent
    W1 = W1 + np.matmul(l1.T, delta_l2)
    W0 = W0 + np.matmul(X.T, delta_l1)
```

The 2-layer neural network implemented in the previous slide