Image Processing & Vision

Lecture 11: Neural Networks

Hak Gu Kim

hakgukim@cau.ac.kr

Immersive Reality & Intelligent Systems Lab (IRIS LAB)

Graduate School of Advanced Imaging Science, Multimedia & Film (GSAIM)

Chung-Ang University (CAU)

29 May 2023

Recap: Image Classification

- The score function, f, is defined to map the pixel values of an image to confidence scores for each class
  - $\mathbf{W}$: weights
  - $\mathbf{b}$: bias vector

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

3072×1 10×1

10×1 10×3072

Image, $\mathbf{x} \in \mathbb{R}^{32 \times 32 \times 3}$

$f(\mathbf{x}, \mathbf{W})$

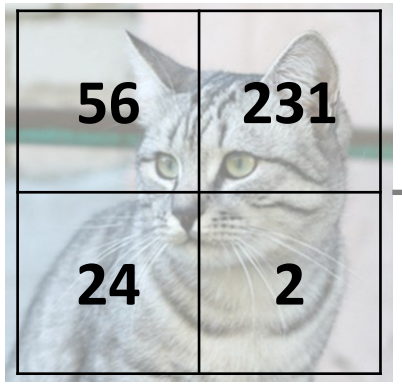
Scores for each class
(**10** classes)

Parameters of weights,
$\mathbf{W} \in \mathbb{R}^{10 \times 3072}$

Recap: Image Classification

- Classification: Cat / Dog / Bird

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$



Input image,
$\mathbf{x} \in \mathbb{R}^{2\times 2}$

Stretch pixels into column vector
(i.e., reshape)

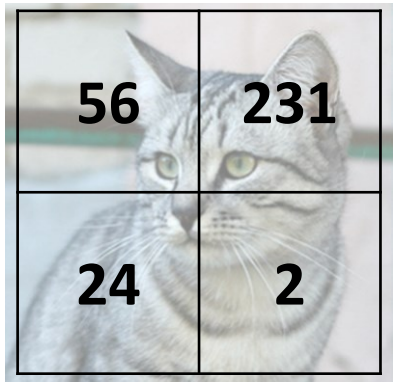
56
231
24
2

$\mathbf{x} \in \mathbb{R}^{4\times 1}$

Recap: Image Classification

- Classification: Cat / Dog / Bird

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$



Input image,
$\mathbf{x} \in \mathbb{R}^{2\times 2}$

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0.0	0.25	0.2	-0.3

$\mathbf{W} \in \mathbb{R}^{3\times 4}$

$\times$

56
231
24
2

$\mathbf{x} \in \mathbb{R}^{4\times 1}$

$+$

1.1
3.2
-1.2

$\mathbf{b} \in \mathbb{R}^{3\times 1}$

$=$

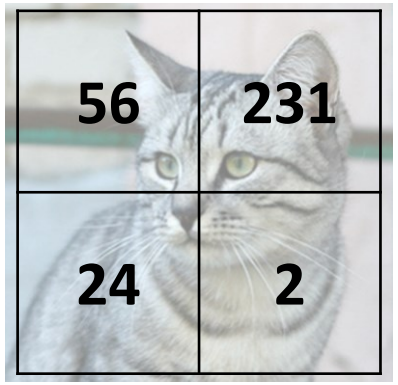
-96.8	Cat
437.9	Dog
61.95	Bird

Score,
$f(\mathbf{x}; \mathbf{W}, \mathbf{b})$

Recap: Image Classification

- Classification: Cat / Dog / Bird

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$



Input image,
$\mathbf{x} \in \mathbb{R}^{2\times 2}$

1.2	1.5	1.1	2.0
-0.5	0.3	0.1	0.0
0.1	0.05	-0.1	-0.2

$\mathbf{W} \in \mathbb{R}^{3\times 4}$

$\times$

56
231
24
2

$\mathbf{x} \in \mathbb{R}^{4\times 1}$

$+$

3.1
1.2
-1.5

$\mathbf{b} \in \mathbb{R}^{3\times 1}$

$=$

447.2	Cat
44.9	Dog
12.85	Bird

Score,
$f(\mathbf{x}; \mathbf{W}, \mathbf{b})$

Topics

- Linear Classifiers for Neural Networks

— Loss Function

— Regularization

— Optimization

***Note:** Many of these slides in this course were adapted from Convolutional Neural Networks for Visual Recognition (Stanford Univ.)

Topics

- Linear Classifiers for Neural Networks
- — Loss Function
- — Regularization
- — Optimization

***Note:** Many of these slides in this course were adapted from Convolutional Neural Networks for Visual Recognition (Stanford Univ.)

Loss Function

- How to choose a good weight, $\mathbf{W}$

— Given a $\mathbf{W}$, we can compute the class scores for an image $\mathbf{x}$

— But how can we actually choose a good $\mathbf{W}$?

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Loss Function

- How to choose a good weight, **W**

— Use **a loss function to quantify** how good a value of **W**

  - **Low loss** value: a **good** weight, **W**
  - **High loss** value: a **bad** weight, **W**

— Find a **W** that minimizes the loss function

: **Optimization**

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Loss Function

- Given a dataset of examples:

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

  - $\mathbf{x}_i$: i-th image
  - y_i: i-th label (integer)
  - N: a total number of images in a dataset

- Loss function for the entire dataset is an average of per-example losses:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i(f(\mathbf{x}_i; \mathbf{W}), y_i)$$

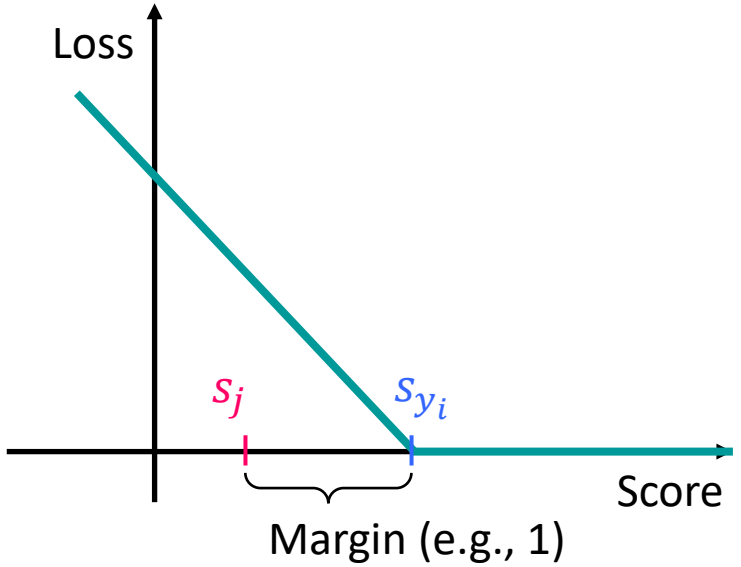
Loss Function: **Multiclass SVM Loss**

- For a multiclass support vector machine (SVM) loss, the score of the correct class, s_{y_i}, should be higher than all the other scores, $s_j (j \neq y_i)$
- We call it **hinge loss function**

— Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores

— Then, the multiclass SVM loss (hinge loss):

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



Loss Function: **Multiclass SVM Loss**

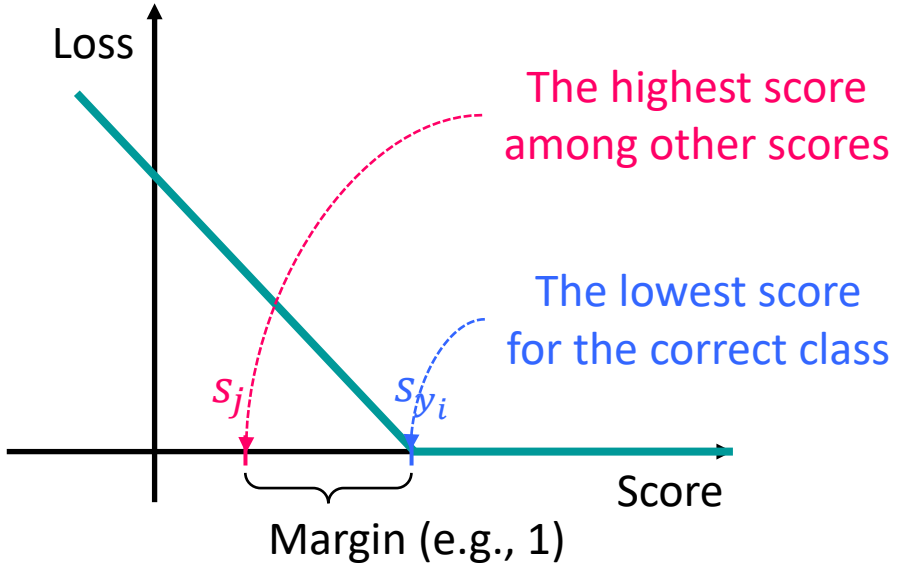
- For a multiclass support vector machine (SVM) loss, the score of the correct class, s_{y_i}, should be higher than all the other scores, $s_j (j \neq y_i)$
- We call it **hinge loss function**

— Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores

— Then, the multiclass SVM loss (hinge loss):

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

$$\mathcal{L}_i = \sum_{j \neq y_i} \begin{cases} 0 & if\ s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & otherwise \end{cases}$$



Loss Function: **Multiclass SVM Loss**

- Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores
- Then, the multiclass SVM loss (hinge loss):

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

	$i = 1$	$i = 2$	$i = 3$
Cat	**3.2**	1.3	2.2
Dog	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**

Loss Function: **Multiclass SVM Loss**

- Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores
- Then, the multiclass SVM loss (hinge loss):

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

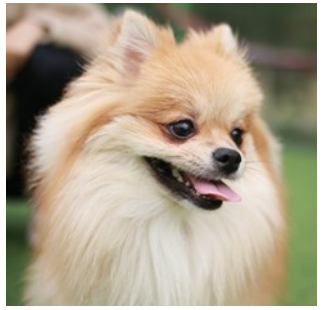
$$= \max(0, 5.1 - 3.2 + 1)$$

$$+ \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$



	$i = 1$	$i = 2$	$i = 3$
Cat	**3.2**	1.3	2.2
Dog	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**
Loss	2.9		

Loss Function: **Multiclass SVM Loss**

- Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores
- Then, the multiclass SVM loss (hinge loss):

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

$$= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)$$

$$= \max(0, -2.6) + \max(0, -1.9)$$

$$= 0 + 0$$

$$= 0$$

	$i = 1$	$i = 2$	$i = 3$
			
Cat	**3.2**	1.3	2.2
Dog	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**
Loss	2.9	0	

Loss Function: **Multiclass SVM Loss**

- Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores
- Then, the multiclass SVM loss (hinge loss):

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

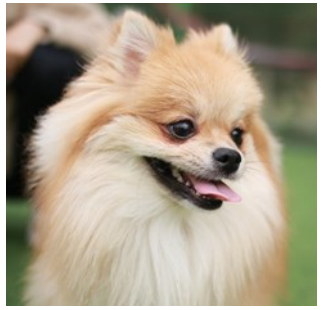
$$= \max(0, 2.2 + 3.1 + 1)$$

$$+ \max(0, 2.5 + 3.1 + 1)$$

$$= \max(0, 6.3) + \max(0, 6.6)$$

$$= 6.3 + 6.6$$

$$= 12.9$$

	$i = 1$ 	$i = 2$ 	$i = 3$ 
Cat	**3.2**	1.3	2.2
Dog	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**
Loss	2.9	0	12.9

Loss Function: **Multiclass SVM Loss**

- Given an example $(\mathbf{x}_i, y_i)$, let $s = f(\mathbf{x}_i; \mathbf{W})$ be scores
- Then, the multiclass SVM loss (hinge loss):

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

	$i = 1$	$i = 2$	$i = 3$
Cat	**3.2**	1.3	2.2
Dog	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**
Loss	2.9	0	12.9

- Final loss for the entire dataset:

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

$$= \frac{1}{3}(2.9 + 0 + 12.9) \approx 5.27$$

Loss Function: **Softmax Function**

- It is often used to normalize the output of a network to **a probability distribution** over predicted output classes

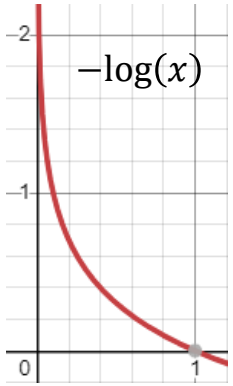
— **Score function:** Unnormalized log probabilities of the classes

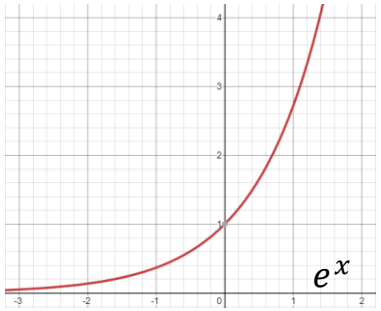
Score function

$$s = f(\mathbf{x}_i; \mathbf{W})$$

Loss function using **softmax function**

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





Loss Function: **Softmax Function**

- It is often used to normalize the output of a network to **a probability distribution** over predicted output classes

— **Score function:** Unnormalized log probabilities of the classes

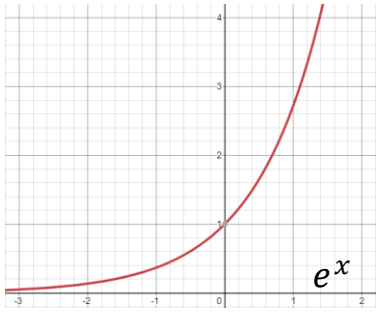
	Score
Cat	**3.2**
Dog	5.1
Bird	-1.7

Score function

$$s = f(\mathbf{x}_i; \mathbf{W})$$

Loss function using **softmax function**

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



Loss Function: **Softmax Function**

- It is often used to normalize the output of a network to **a probability distribution** over predicted output classes

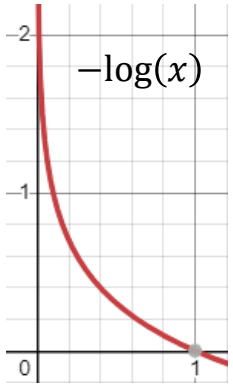
— **Score function:** Unnormalized log probabilities of the classes

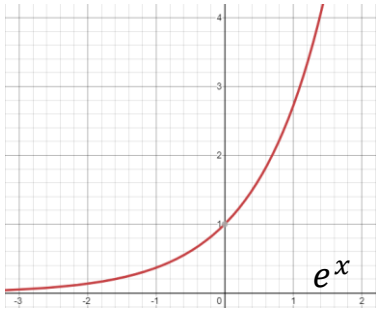
Score function

$$s = f(\mathbf{x}_i; \mathbf{W})$$

Loss function using **softmax function**

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



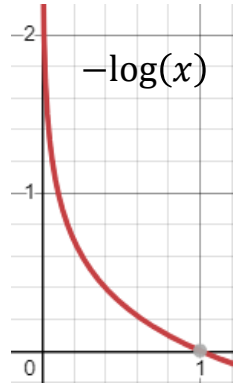


Cat	**3.2**	Exp, e →	**24.5**
Dog	5.1		164.0
Bird	-1.7		0.18
	Score		**Unnormalized probabilities**

Loss Function: **Softmax Function**

- It is often used to normalize the output of a network to **a probability distribution** over predicted output classes

— **Score function:** Unnormalized log probabilities of the classes

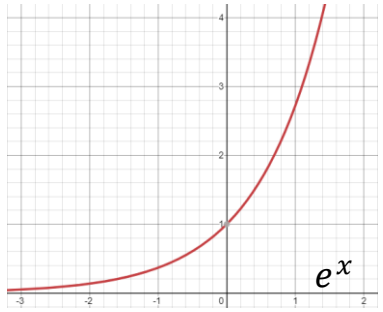


Score function

$$s = f(\mathbf{x}_i; \mathbf{W})$$

Loss function using **softmax function**

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

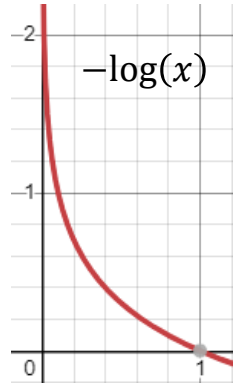


	Score	Exp, e	Unnormalized probabilities	Normalize	Softmax
Cat	**3.2**		**24.5**		**0.13**
Dog	5.1	→	164.0	→	0.87
Bird	-1.7		0.18		0.00

Loss Function: **Softmax Function**

- It is often used to normalize the output of a network to **a probability distribution** over predicted output classes

— **Score function:** Unnormalized log probabilities of the classes

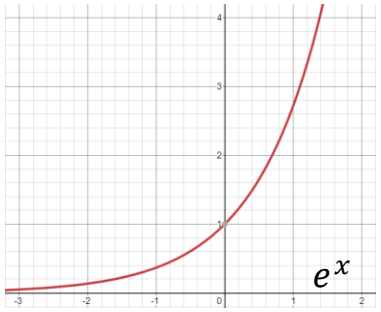


Score function

$$s = f(\mathbf{x}_i; \mathbf{W})$$

Loss function using **softmax function**

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



	Score		Unnormalized probabilities		Softmax	
Cat	**3.2**	Exp, e	**24.5**	Normalize	**0.13**	**=24.5/188.68**
Dog	5.1	→	164.0	→	0.87	=164.0/188.68
Bird	-1.7		0.18		0.00	=0.18/188.68

Loss Function: **Softmax Function**

- It is often used to normalize the output of a network to **a probability distribution** over predicted output classes

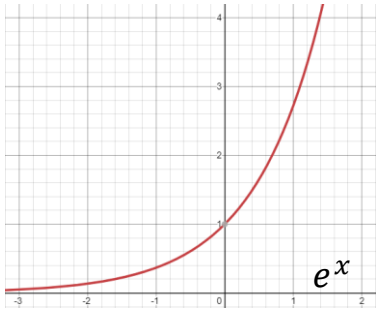
— **Score function:** Unnormalized log probabilities of the classes

Score function

$$s = f(\mathbf{x}_i; \mathbf{W})$$

Loss function using **softmax function**

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



Cat	**3.2**	**24.5**	**0.13**
Dog	5.1	164.0	0.87
Bird	-1.7	0.18	0.00
	Score	**Unnormalized probabilities**	**Softmax**

Exp, e →

Normalize →

$$= \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

- $\mathcal{L}_i = -\log(0.13)$
  $= 0.89$

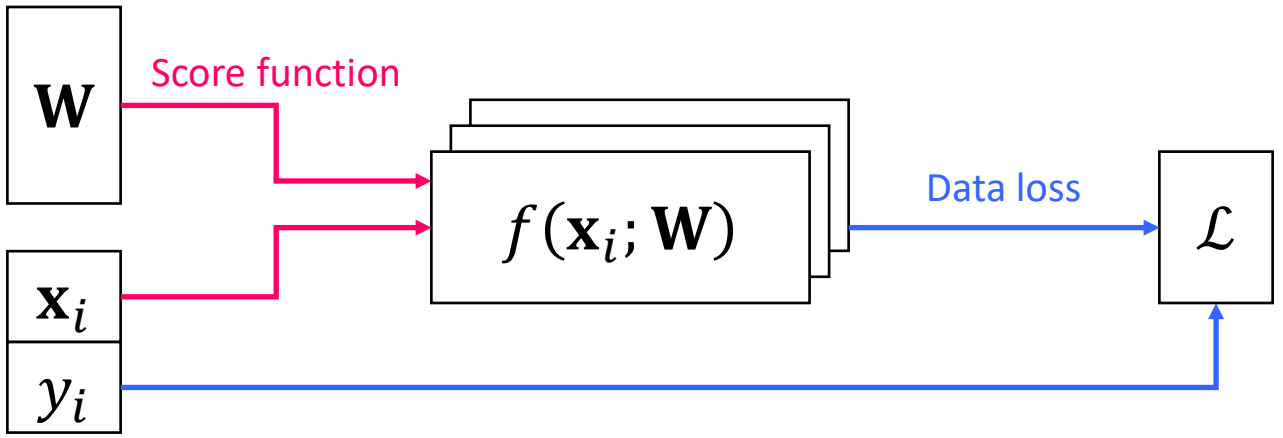
Summary: **Loss Function**

- Loss functions quantify the preferences (in general, negativeness)

— Score function: $s = f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ (i.e., linear classifier)

— Loss function:

- Multiclass SVM function: $\mathcal{L}_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
- Softmax function: $\mathcal{L}_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$

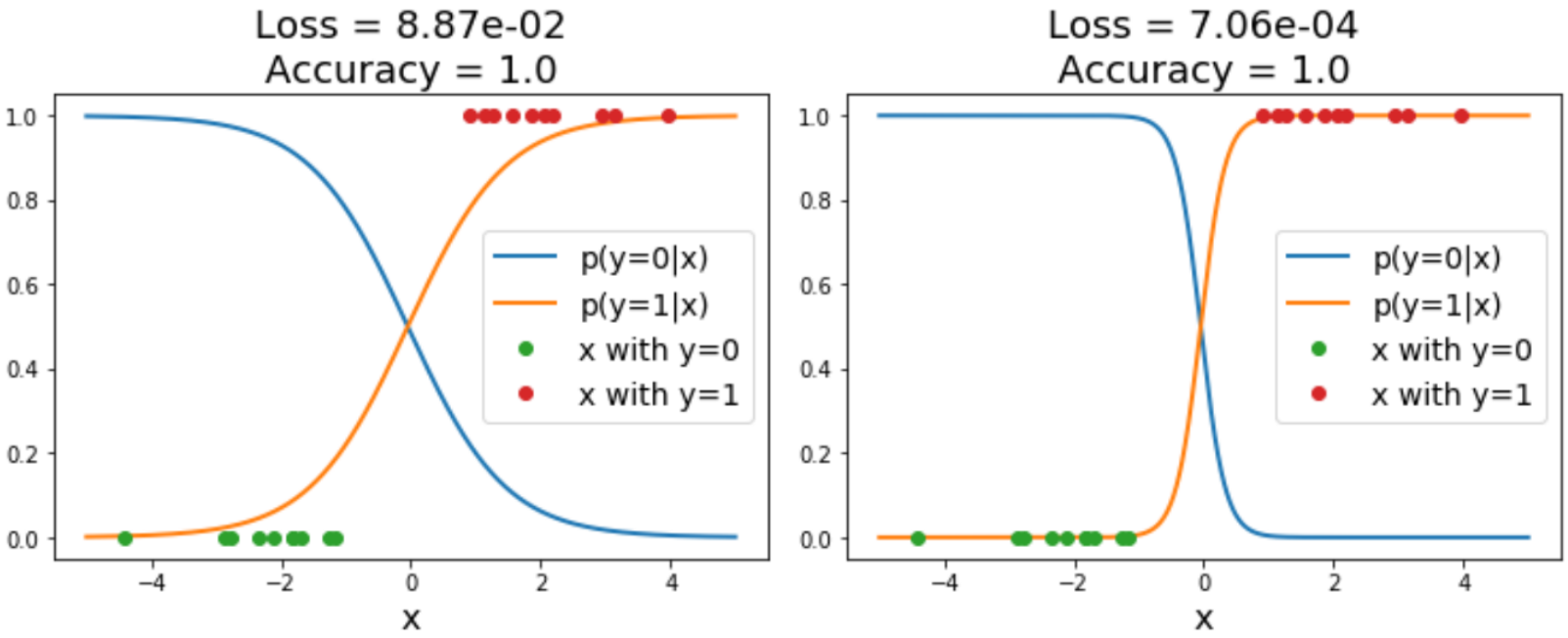


Topics

- Linear Classifiers for Neural Networks
  - Loss Function
  - Regularization
  - Optimization

Overfitting

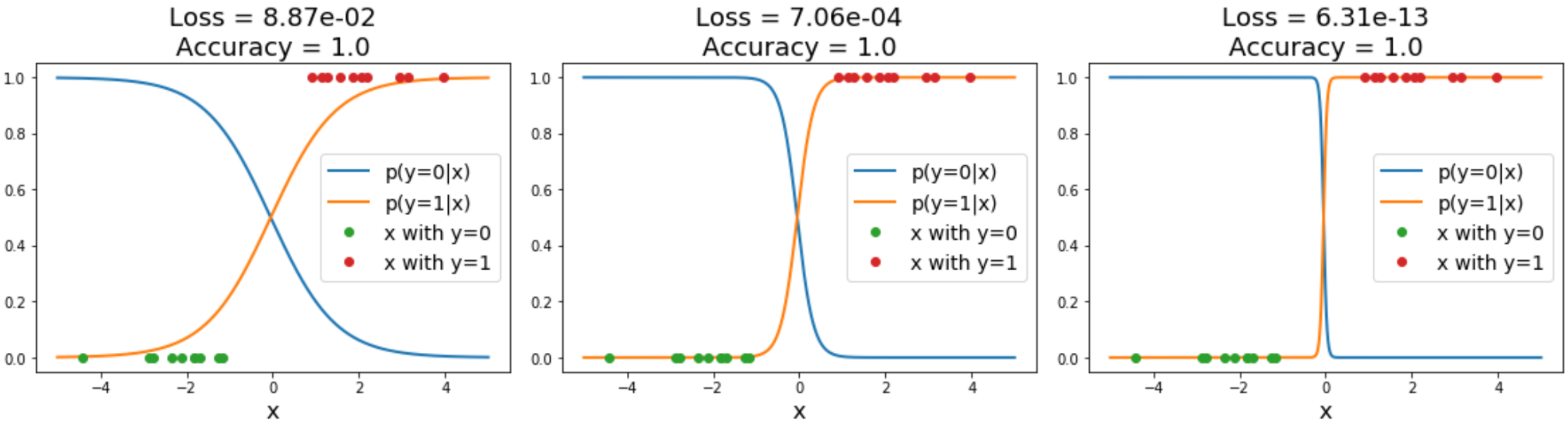
- A model (classifier) is **overfit** when it performs **too well on the training dataset**, and has **poor performances for unseen data** (e.g., test dataset)



Both models have perfect accuracy on training dataset

Overfitting

- A model (classifier) is **overfit** when it performs **too well on the training dataset**, and has **poor performances for unseen data** (e.g., test dataset)

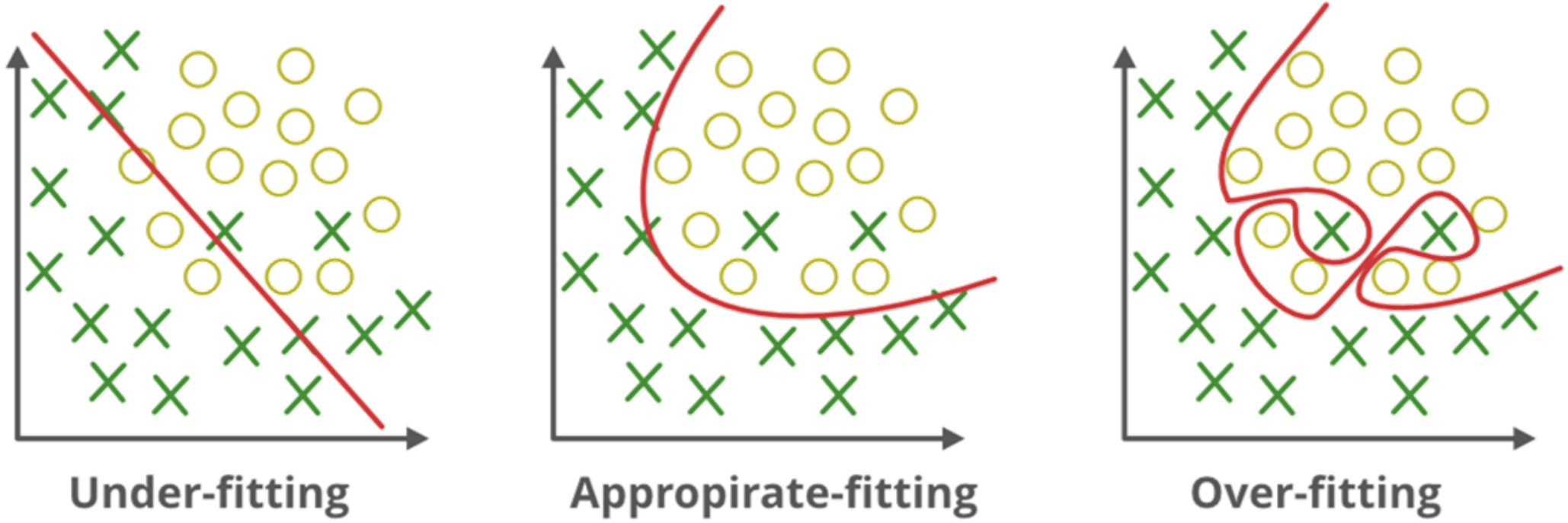


Both models have perfect accuracy on training dataset

Very low loss, but unnatural cliff between training points

Overfitting

- Suppose that we have a dataset and a set of parameters $\mathbf{W}$ that correctly classify every example
- The issue is that this set of $\mathbf{W}$ is **not** necessarily **unique**: there might be many similar $\mathbf{W}$ that correctly classify the examples



https://www.geeksforgeeks.org/regularization-in-machine-learning/

Regularization: Beyond Training Error

- We wish to encode some preference for a certain set of weights $\mathbf{W}$ over others **to remove this ambiguity**

— **Data loss:** Model predictions should match training data

— **Regularization:** It prevents the model from doing too well on training data

  - ℓ_1 regularization: $R(\mathbf{W}) = \sum_{k,l} |\mathbf{W}_{k,l}|$
  - ℓ_2 regularization: $R(\mathbf{W}) = \sum_{k,l} \mathbf{W}_{k,l}^2$

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_i\left(f(\mathbf{x}_i; \mathbf{W}), y_i\right) + \lambda R(\mathbf{W})$$

Data loss **Regularization**

Example: **Regularization**

- We wish to encode some preference for a certain set of weights $\mathbf{W}$ over others **to remove this ambiguity**

— **Data loss:** Model predictions should match training data

— **Regularization:** It prevents the model from doing too well on training data

  - ℓ_1 regularization: $R(\mathbf{W}) = \sum_{k,l} |\mathbf{W}_{k,l}|$
  - ℓ_2 regularization: $R(\mathbf{W}) = \sum_{k,l} \mathbf{W}_{k,l}^2$

- Examples of ℓ_2 **regularization:**

$\mathbf{x} = [1, 1, 1, 1]^T$ $\quad \mathbf{W}_1 = [1, 0, 0, 0]^T$ $\quad R(\mathbf{W}_1) = 1^2 = 1$

$\mathbf{W}_2 = [0.25, 0.25, 0.25, 0.25]^T$ $\quad R(\mathbf{W}_2) = 0.25^2 \times 4 = 0.25$

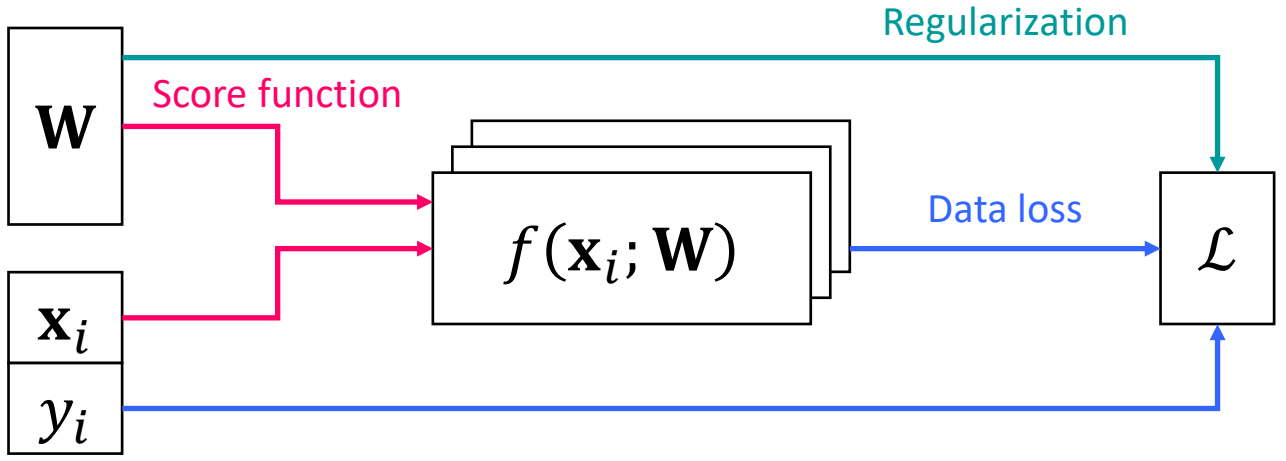
Summary: **Regularization**

- A model (classifier) is **overfit** when it performs **too well on the training dataset**, and has **poor performances for unseen data** (e.g., test dataset)
- To deal with that, the loss function consists of **data loss** to fit the training data and **regularization** to prevent overfitting

— **Data loss:** Model predictions should match training data

— **Regularization:** It prevents the model from doing too well on training data

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_i\left(f(\mathbf{x}_i;\mathbf{W}), y_i\right) + \lambda R(\mathbf{W})$$



Topics

- Linear Classifiers for Neural Networks
  - Loss Function
  - Regularization
  - Optimization

Optimization

- Two key components in context of the image classification task:

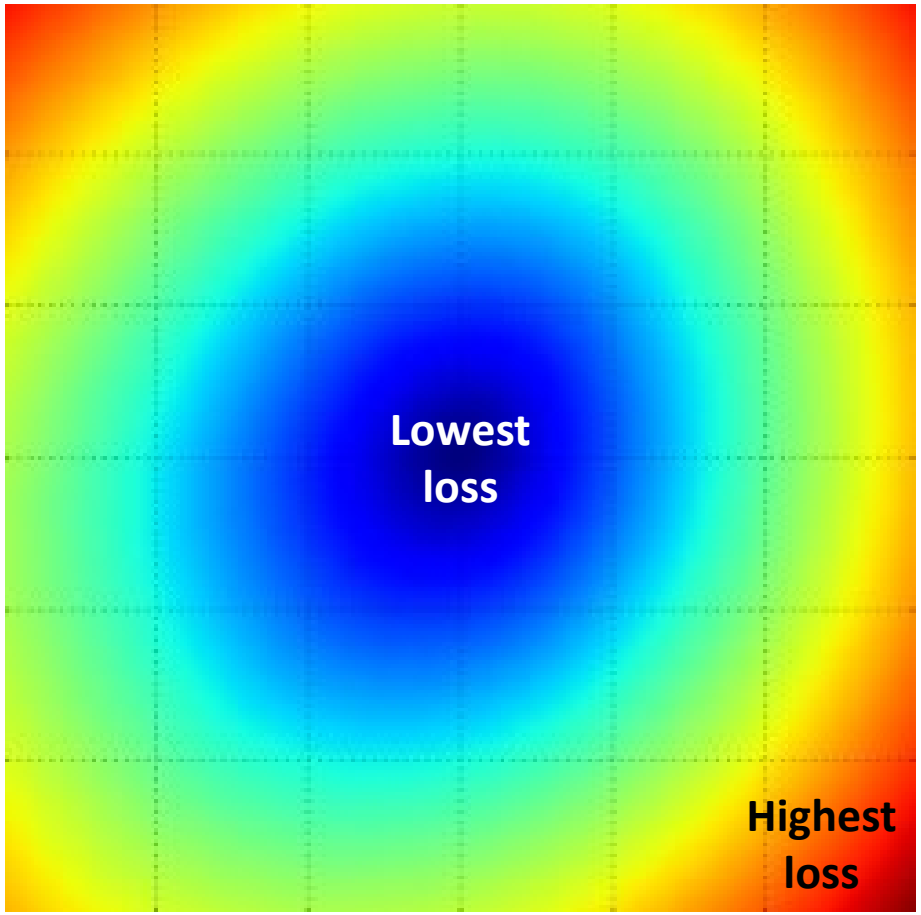
— **A score function** mapping the raw image pixels to class scores

— **A loss function** that measured the quality of a particular set of parameters based on how well the induced scores agreed with the ground truth labels in the training data

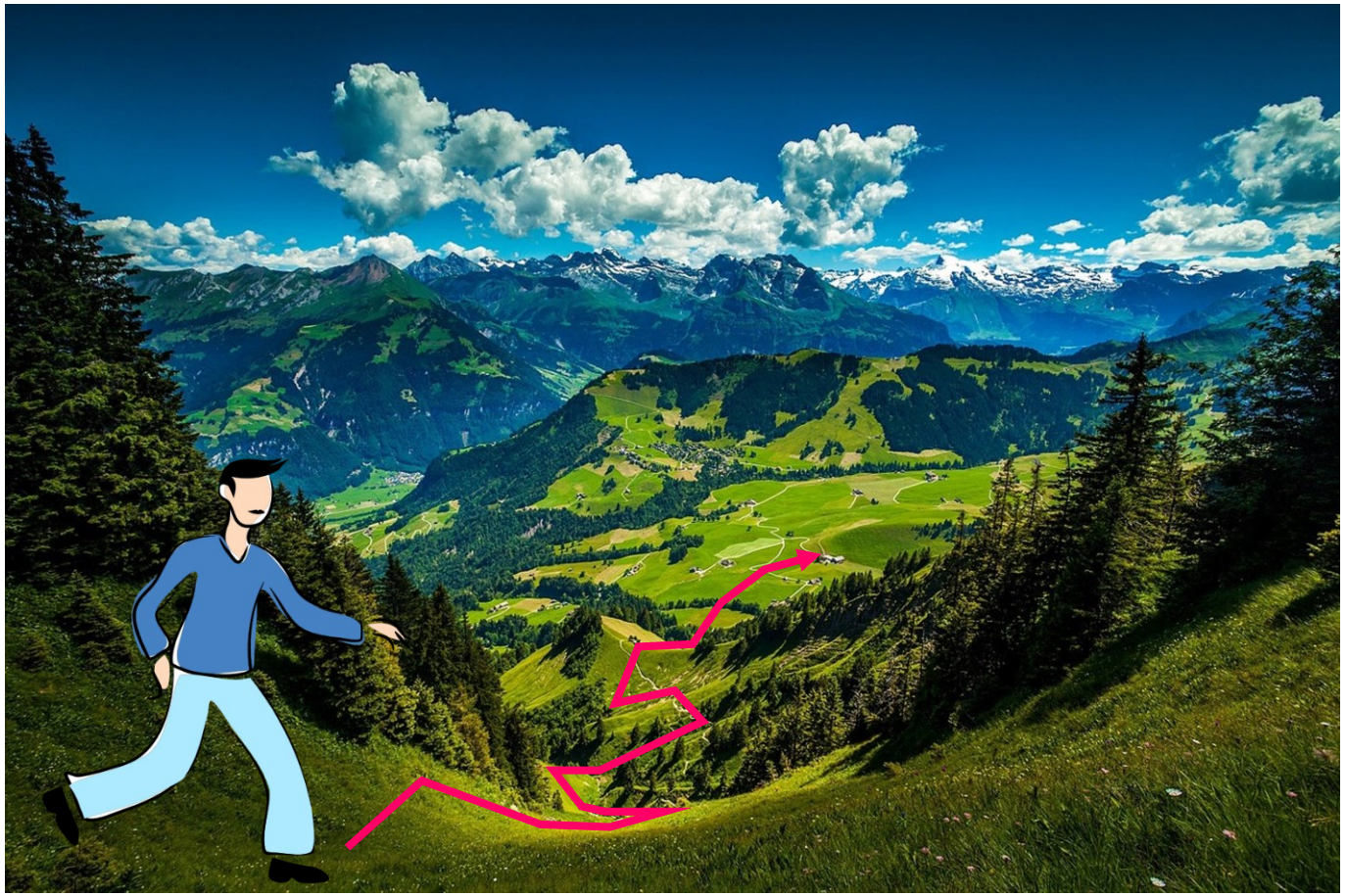
- **Optimization** is the process of **finding** the set of parameters $\mathbf{W}$ that **minimize the loss function**

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \mathcal{L}(\mathbf{W})$$

Visualization of **Loss Function**



Loss function landscape for the Multiclass SVM (without regularization) for a hundred examples in CIFAR-10



Optimization: **Random Search**

- The first idea that may come to mind is to simply **try out many different random weights** and keep track of what works best: **Very bad**

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in range(1000):
  W = np.random.randn(10, 3073) * 0.0001
  loss = L(X_train, Y_train, W) # get the
  if loss < bestloss: # keep track of the
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
```

: Random number generation for weights **W**

: Loss calculation with the randomly generated **W**

: Best loss is updated if the current loss is lower than the best loss

Optimization: **Random Search**

- In the code in the previous slide, we see that we tried out several random weight vectors $\mathbf{W}$, and some of them work better than others
- Take the **best weights $\mathbf{W}$** found by this search and **try it out on the test set:** With the best $\mathbf{W}$ this gives an accuracy of about **15.5% (SOTA ≈ 95%)**

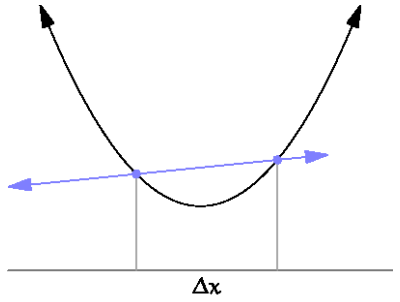
```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

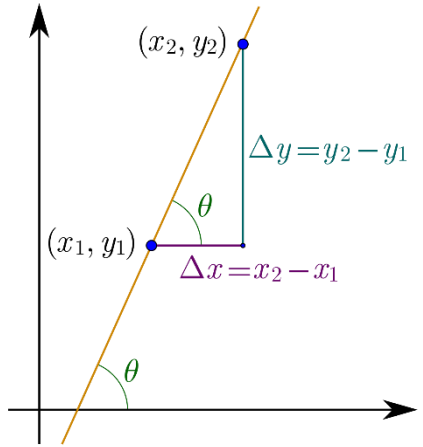
Optimization: **Following the Gradient**

- The best direction will be related to **the gradient of the loss function**
- This approach roughly corresponds to **feeling the slope** of the hill below our feet and **stepping down the direction** that feels steepest

- In 1-dimension, the derivative of a function gives the slope:

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





- The slope in any direction is the **dot product of the direction** with the gradient and the direction of steepest descent is the **negative gradient**

Optimization: **Following the Gradient**

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 dL/dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,
?,…]

Optimization: **Following the Gradient**

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 dL/dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

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

?,
?,...]

Optimization: **Following the Gradient**

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 dL/dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,...]

Optimization: **Following the Gradient**

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 dL/dW:

[-2.5,
0.6,
?,
?,

(1.25353 - 1.25347)/0.0001
= 0.6

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

?,…]

Optimization: **Following the Gradient**

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 (third dim)**:**

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

loss 1.25347

gradient dL/dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,…]

Optimization: **Following the Gradient**

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 (third dim):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]
loss 1.25347

gradient dL/dW:

[-2.5,
0.6,
0.0,
?,
?,

(1.25347 - 1.25347)/0.0001
= 0.0

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

Optimization: **Following the Gradient**

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 (third dim)**:**

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

loss 1.25347

gradient dL/dW:

[-2.5,
0.6,
0.0,
?,
?,

Numeric Gradient:
- Slow: O(#dimensions)
- Approximate

Efficiency of **Computing the Numerical Gradient**

- Evaluating the numerical gradient has **complexity linear** in the number of parameters
- This problem only gets worse, since modern neural networks can easily have **tens of millions** of parameters. Clearly, this strategy is **not scalable**

- Computing the gradient analytically **with Calculus:**

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i\left(f(\mathbf{x}_i;\mathbf{W}), y_i\right) + \lambda R(\mathbf{W}) \xrightarrow{\text{Calculus}} \nabla_{\mathbf{W}}\mathcal{L}$$

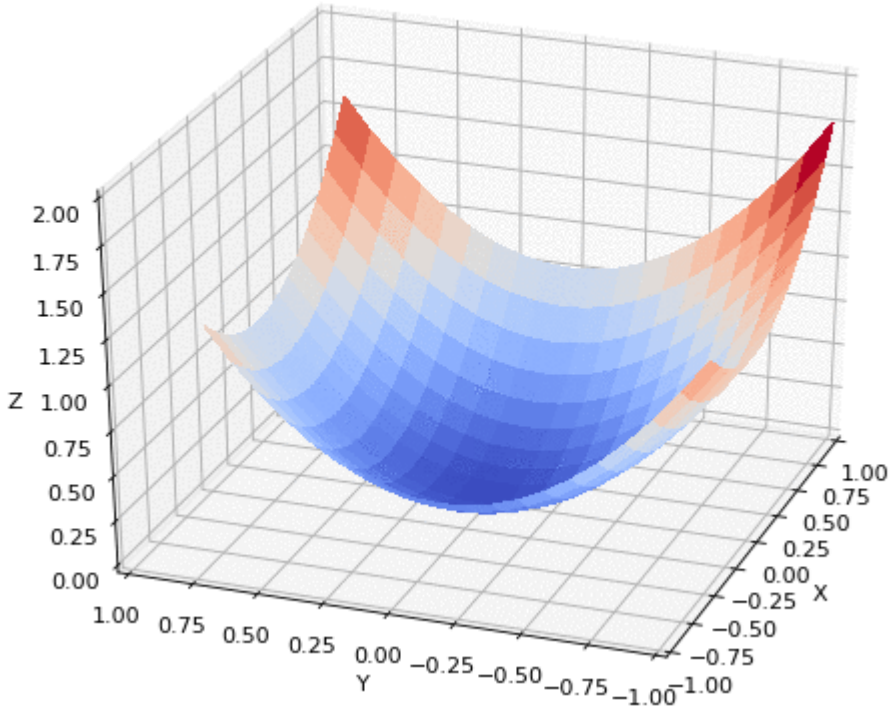
: The analytic gradient

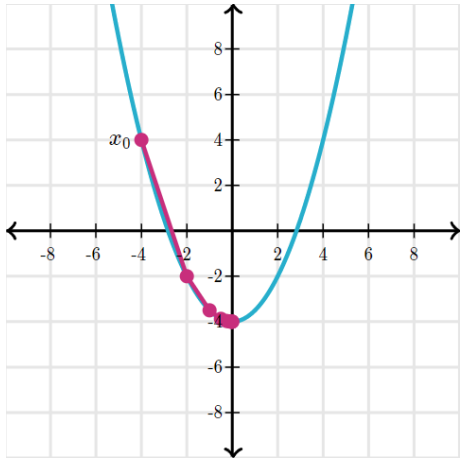
Following the Gradient: **Gradient Descent Method**

- Gradient descent method is the procedure of **repeatedly evaluating the gradient** and then performing a parameter **update**

— α: step size or learning rate

$$\mathbf{W}_{n+1} = \mathbf{W}_n - \alpha \nabla_{\mathbf{W}} \mathcal{L}$$





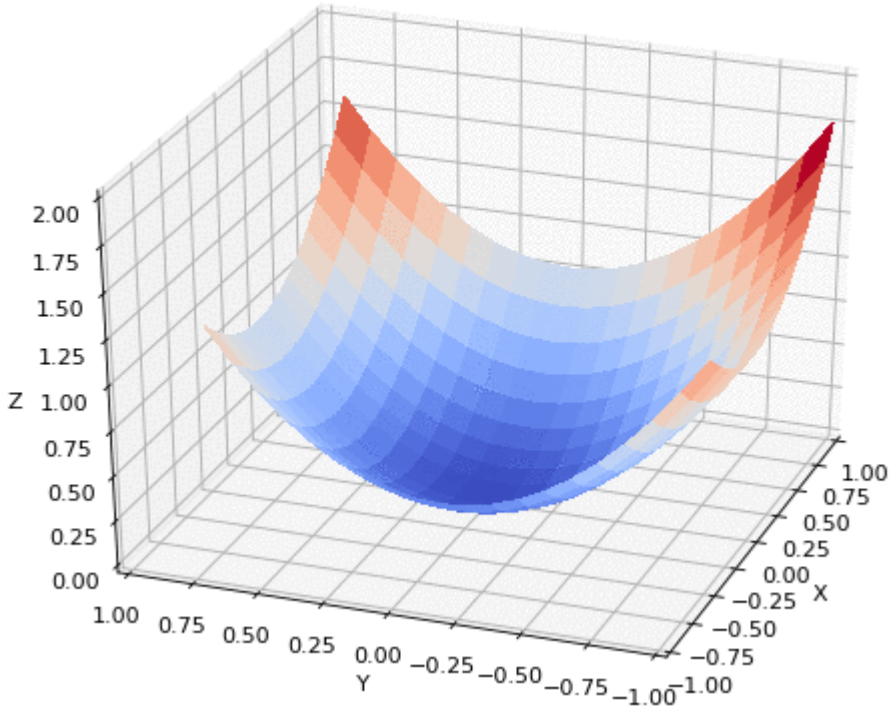
Proper learning rate

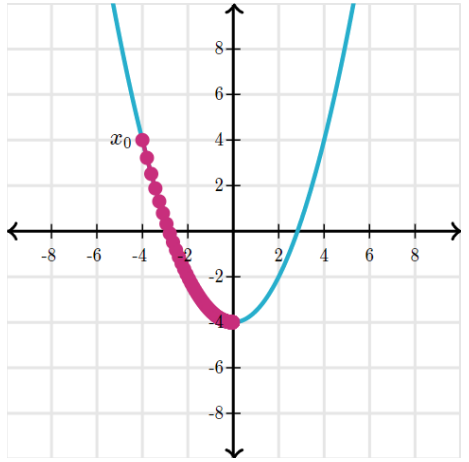
Following the Gradient: **Gradient Descent Method**

- Gradient descent method is the procedure of **repeatedly evaluating the gradient** and then performing a parameter **update**

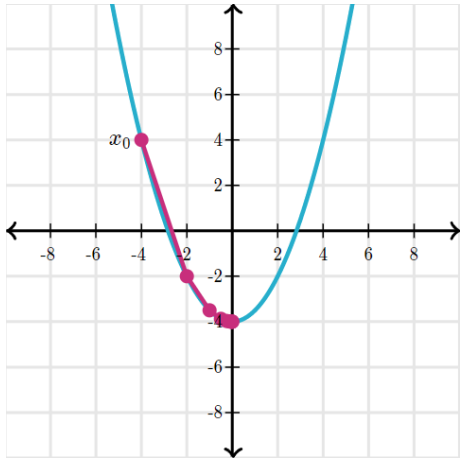
— α: step size or learning rate

$$\mathbf{W}_{n+1} = \mathbf{W}_n - \alpha \nabla_{\mathbf{W}} \mathcal{L}$$

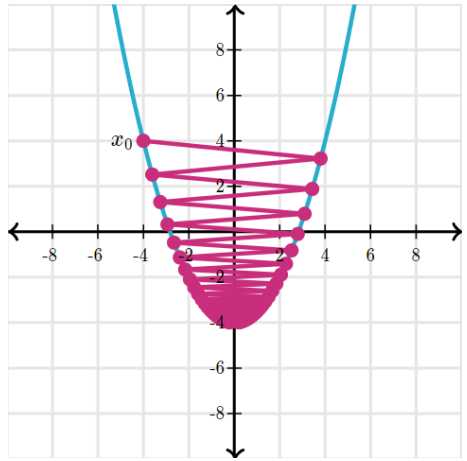


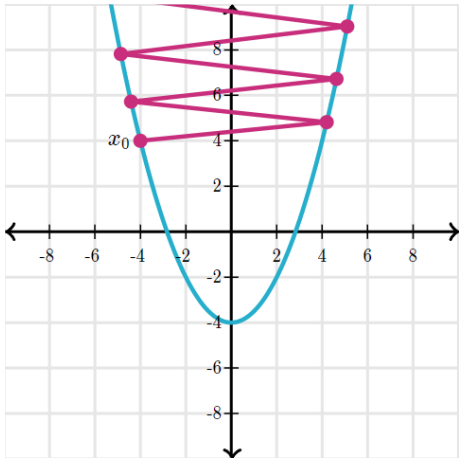


Learning rate is too small



Proper learning rate





Learning rate is too large

Summary: **Optimization**

- The intuition of the loss function as a high-dimensional optimization landscape in which we are trying **to reach the bottom**

- **The gradient** of a function gives **the steepest ascent direction**:

— Numerical gradient: simple but it is approximate and expensive to compute

— Analytic gradient: exact, fast to compute but more error-prone

- The parameter update requires a tricky setting of the step size (or the learning rate) that must be set just right
  - If it is too low the progress is steady but slow
  - If it is too high the progress can be faster, but more risky

Neural Networks

- **Linear Classifier:** $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$

— Learnable parameters

  - $\mathbf{W} \in \mathbb{R}^{C \times D}$
  - $\mathbf{b} \in \mathbb{R}^{C \times 1}$

- **2-layer Neural Network:** $f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$

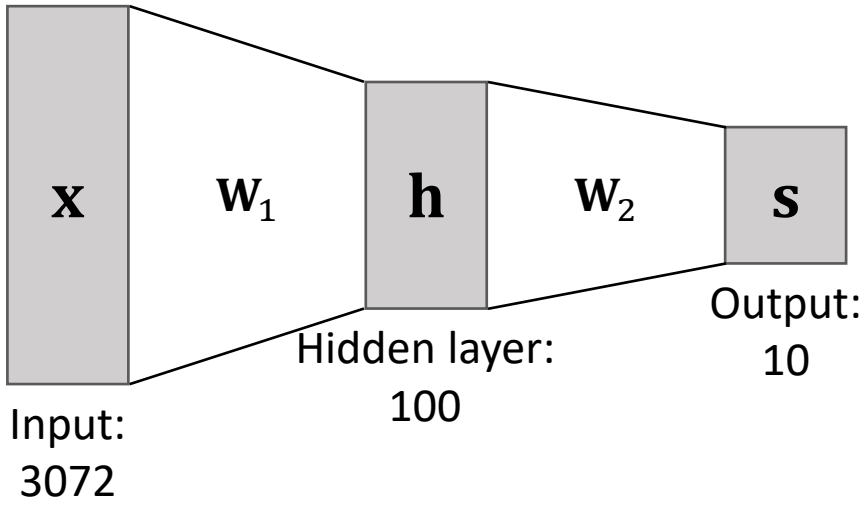
— Learnable parameters

  - $\mathbf{W}_1 \in \mathbb{R}^{H \times D}$, $\mathbf{W}_2 \in \mathbb{R}^{C \times H}$
  - $\mathbf{b}_1 \in \mathbb{R}^{H \times 1}$, $\mathbf{b}_2 \in \mathbb{R}^{C \times 1}$

Neural Networks: **Multi-Layer Perceptron (MLP)**

- Input: $\mathbf{x} \in \mathbb{R}^{D \times 1}$ $where$ $D = 3072$
- Output: $f(\mathbf{x}) \in \mathbb{R}^{C \times 1}$ $where$ $C = 10$
- **Linear Classifier:** $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- **2-layer Neural Network:** $f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$

Element of (i, j) of $\mathbf{W}_1$ gives the effect on h_i from x_j

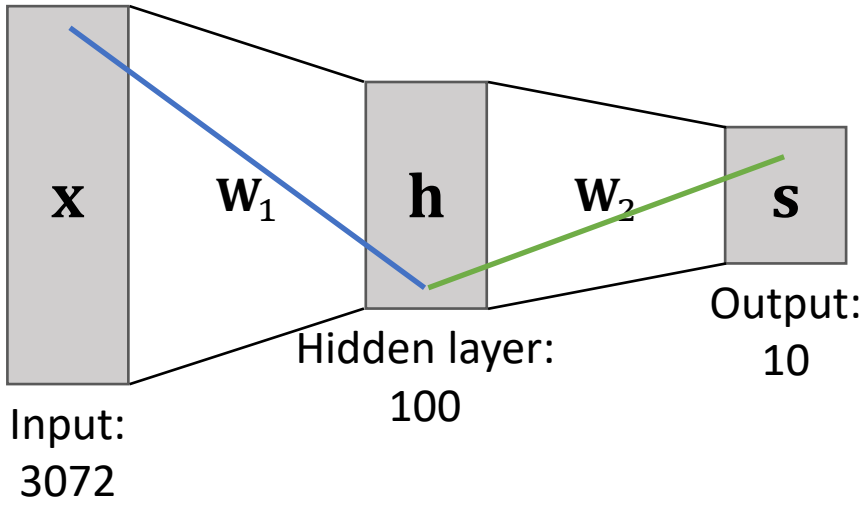


Element of (i, j) of $\mathbf{W}_2$ gives the effect on s_i from h_j

Neural Networks: **Multi-Layer Perceptron (MLP)**

- Input: $\mathbf{x} \in \mathbb{R}^{D \times 1}$ $where\ D = 3072$
- Output: $f(\mathbf{x}) \in \mathbb{R}^{C \times 1}$ $where\ C = 10$
- **Linear Classifier:** $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- **2-layer Neural Network:** $f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$

Element of (i, j) of $\mathbf{W}_1$ gives the effect on h_i from x_j

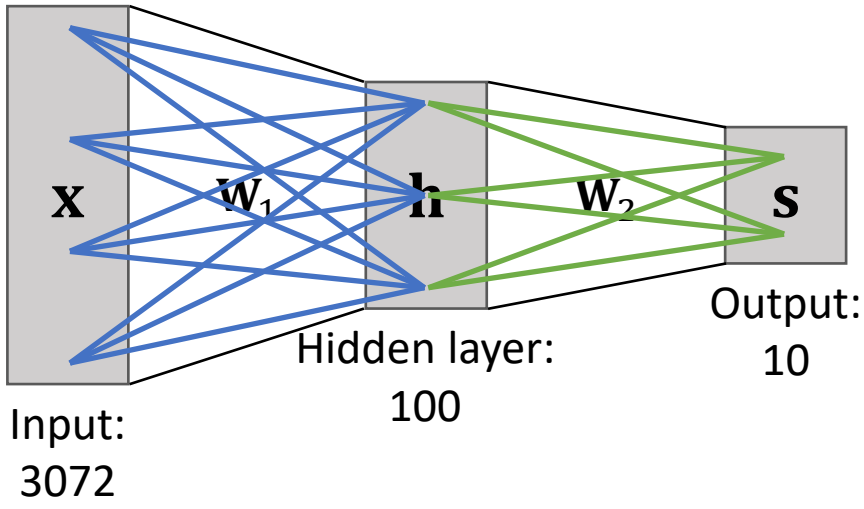


Element of (i, j) of $\mathbf{W}_2$ gives the effect on s_i from h_j

Neural Networks: **Multi-Layer Perceptron (MLP)**

- Input: $\mathbf{x} \in \mathbb{R}^{D \times 1}$ *where* $D = 3072$
- Output: $f(\mathbf{x}) \in \mathbb{R}^{C \times 1}$ *where* $C = 10$
- **Linear Classifier:** $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- **2-layer Neural Network:** $f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$

All elements of $\mathbf{x}$ affect all elements of $\mathbf{h}$



All elements of $\mathbf{h}$ affect all elements of $\mathbf{s}$

Fully-connected neural network (MLP)

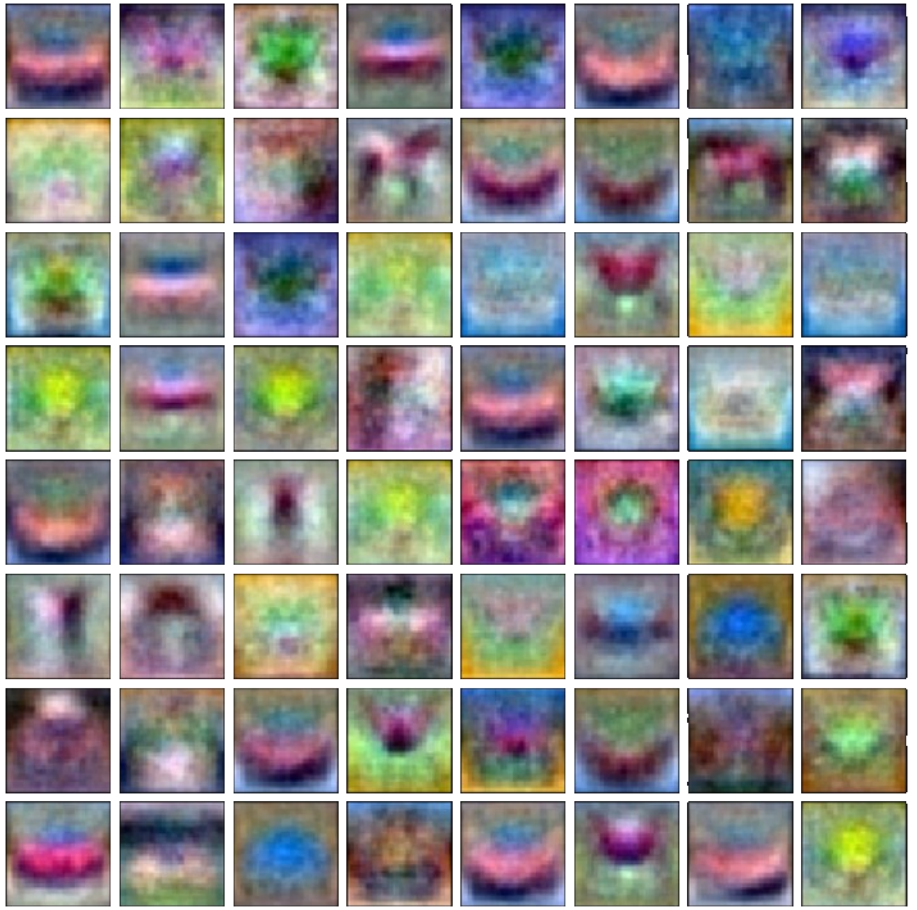
Linear Classifier vs. 2-layer Neural Network

- **Linear Classifier**

— One template per class

- **2-layer Neural Network**

— Different templates per class



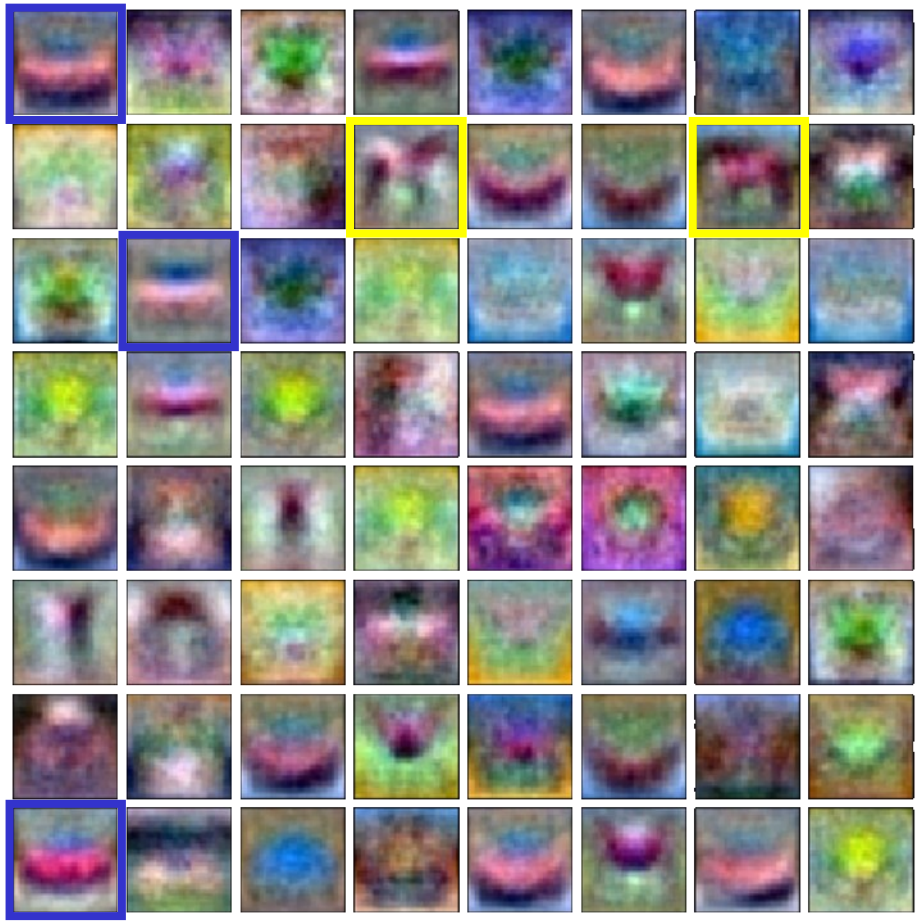
Linear Classifier vs. 2-layer Neural Network

- **Linear Classifier**

— One template per class

- **2-layer Neural Network**

— Different templates per class



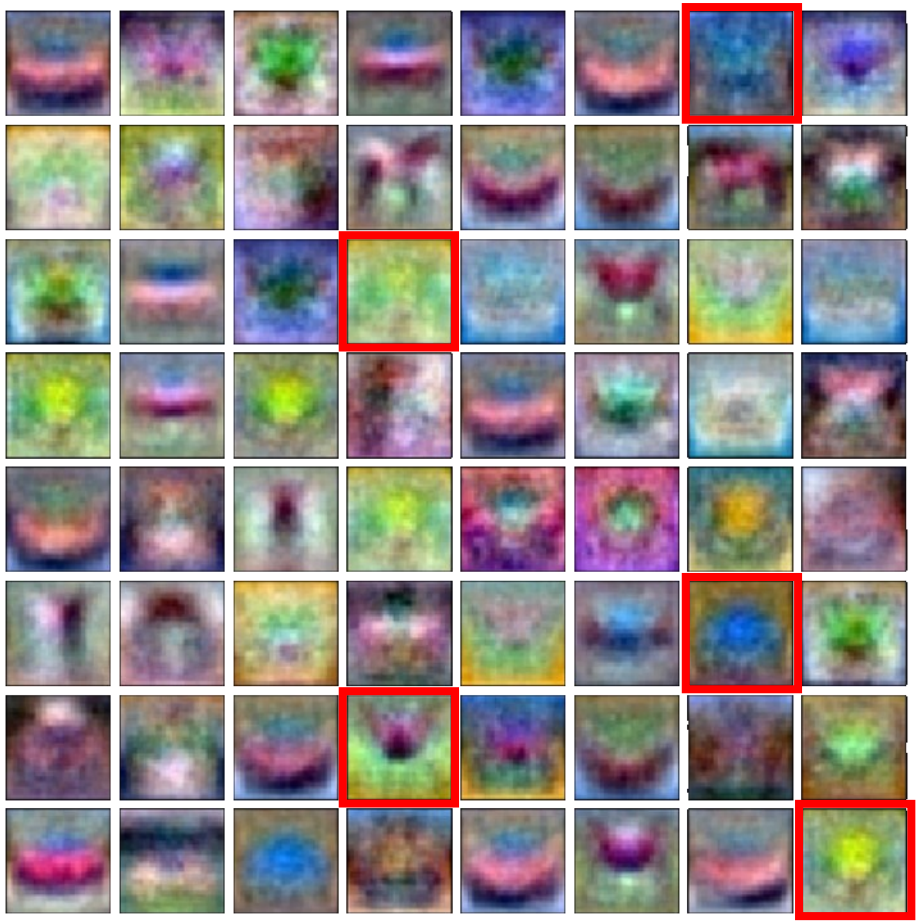
Linear Classifier vs. 2-layer Neural Network

- **Linear Classifier**

— One template per class

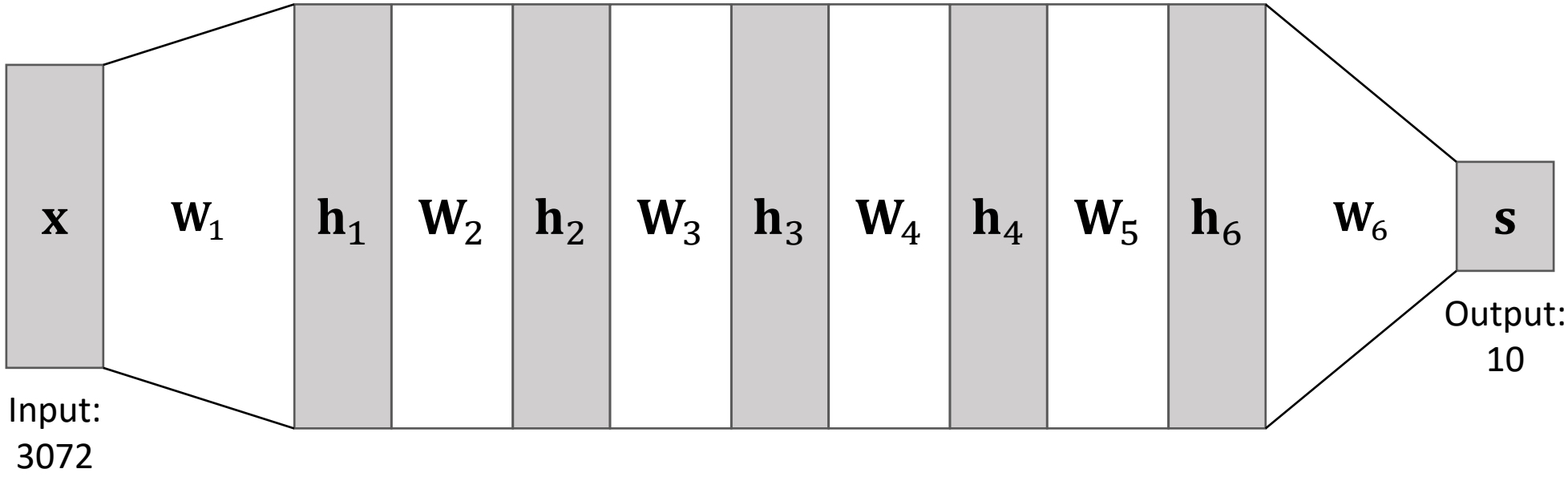
- **2-layer Neural Network**

— Most templates **not interpretable**



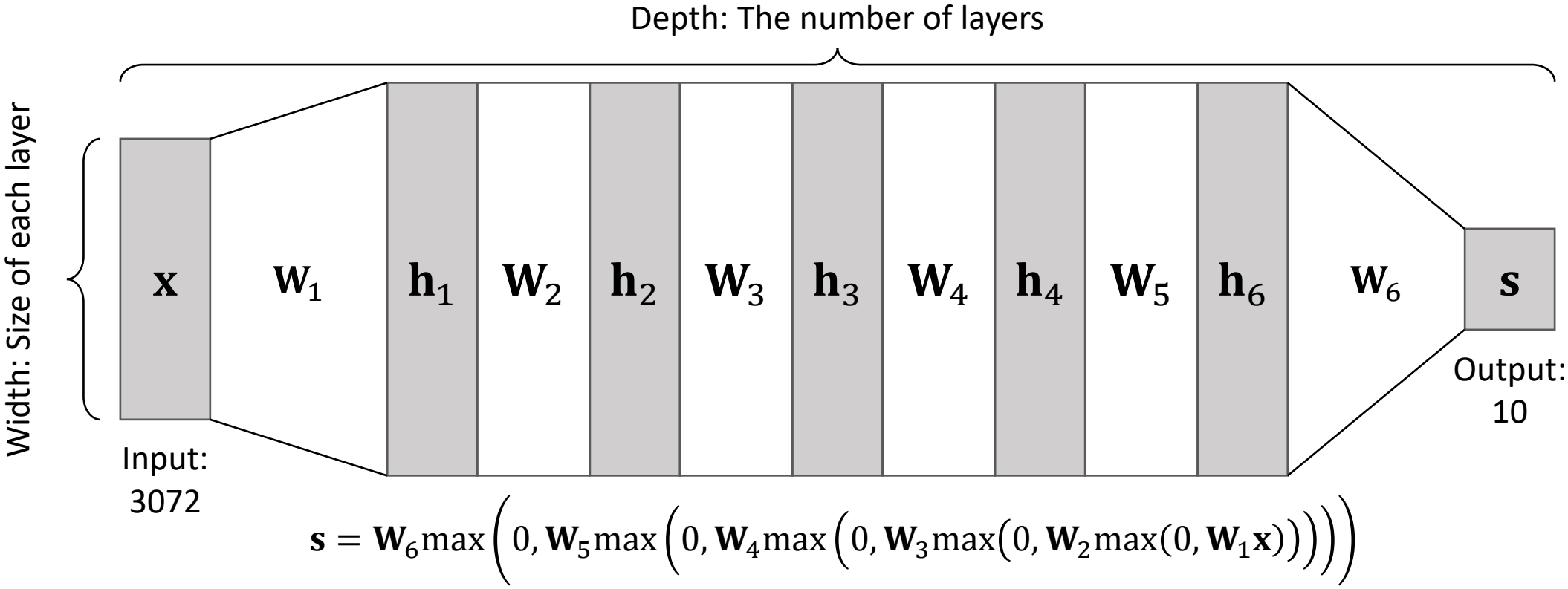
Deep Neural Networks

- **A deep neural network (DNN)** is an artificial neural network (ANN) with **multiple layers between the input and output layers**
  - The components of DNN functioning **similar to the human brains** and can be trained like any other ML algorithm



Deep Neural Networks

- **A deep neural network (DNN)** is an artificial neural network (ANN) with **multiple layers between the input and output layers**
  - The components of DNN functioning **similar to the human brains** and can be trained like any other ML algorithm

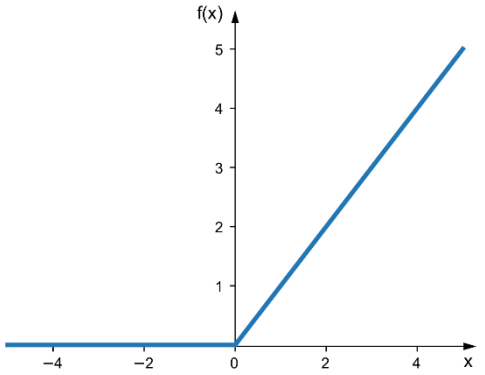


Activation Functions in ANNs

- In ANNs, **the activation function** of a node defines **the output of that node** given an input or set of inputs. It is called **nonlinearities**

— The function ReLU (Rectified Linear Unit): $ReLU(z) = \max(0, z)$

$$f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$



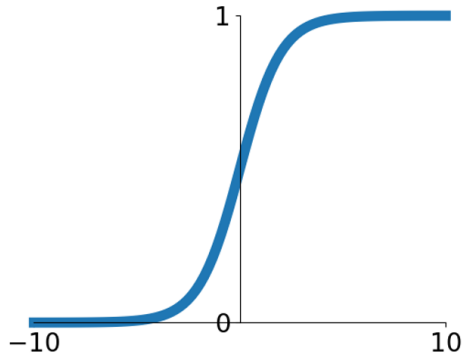
- If we build a neural network without activation function, it ends up with a linear classifier

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{W}_2(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} + (\mathbf{W}_2 \mathbf{b}_1 + \mathbf{b}_2) \end{aligned}$$

Activation Functions in ANNs

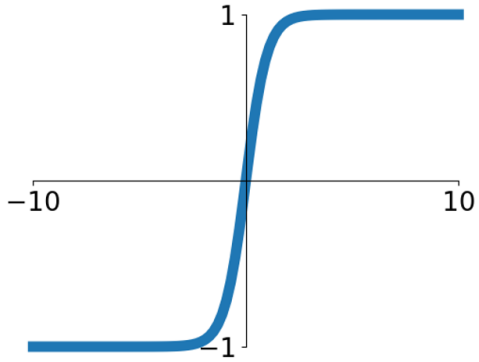
- **Sigmoid**

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



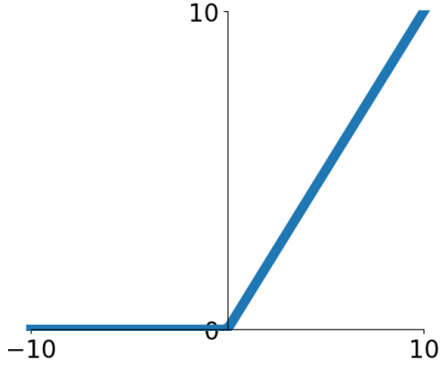
- **tanh**

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



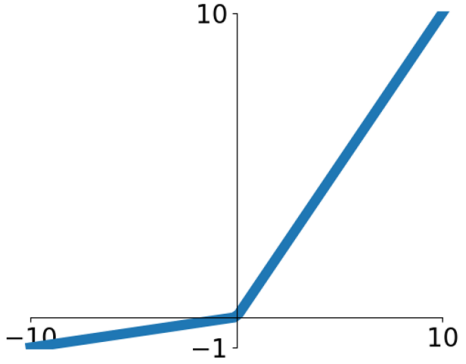
- **ReLU**

$$\max(0, x)$$



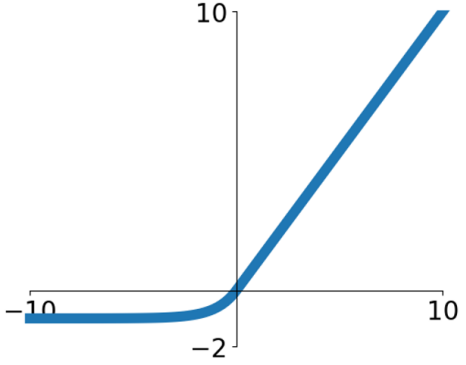
- **Leaky ReLU**

$$\max(0.2x, x)$$



- **ELU**

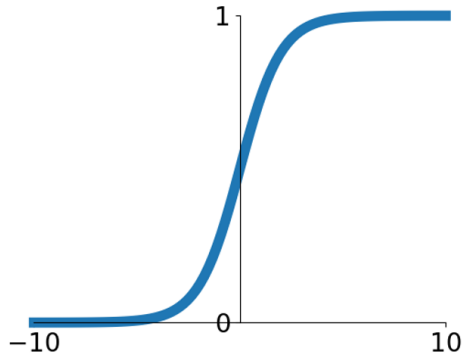
$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$$



Activation Functions in ANNs

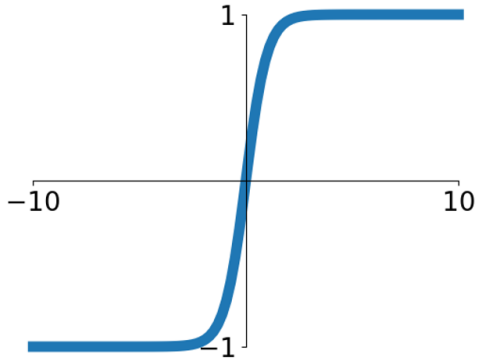
- **Sigmoid**

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



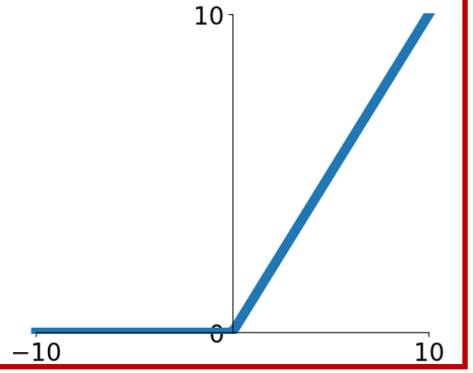
- **tanh**

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



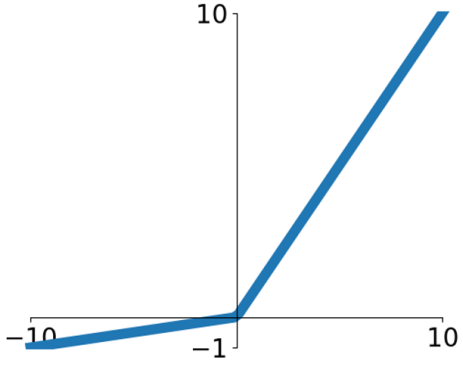
- **ReLU**

$$\max(0, x)$$



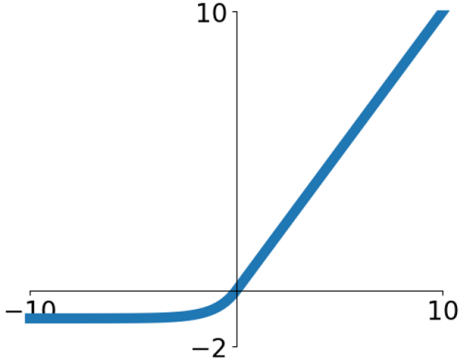
- **Leaky ReLU**

$$\max(0.2x, x)$$



- **ELU**

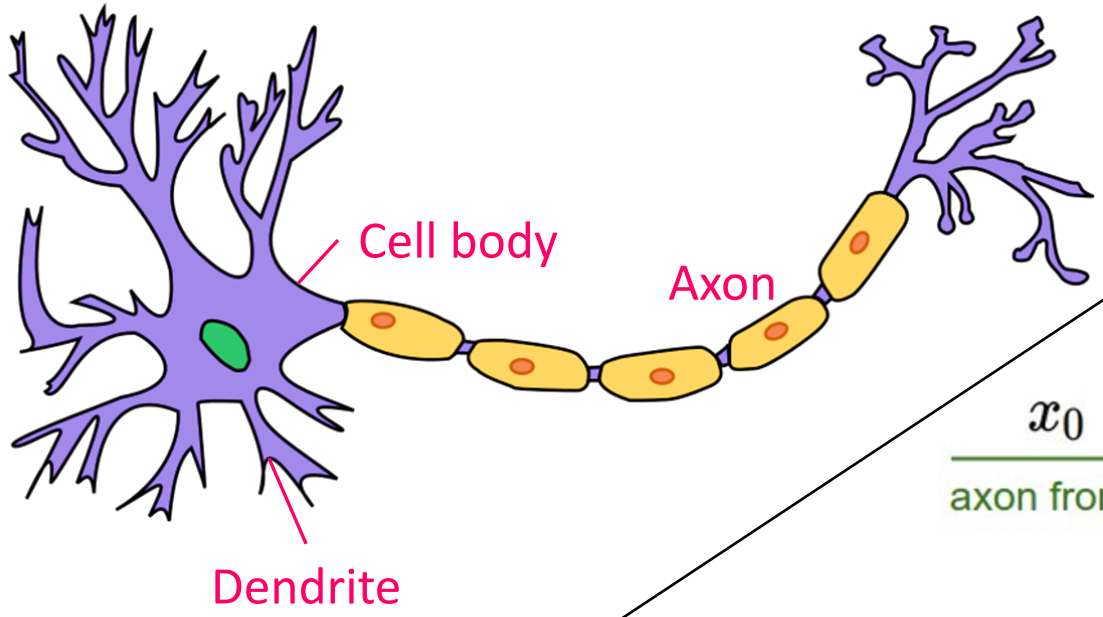
$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$$

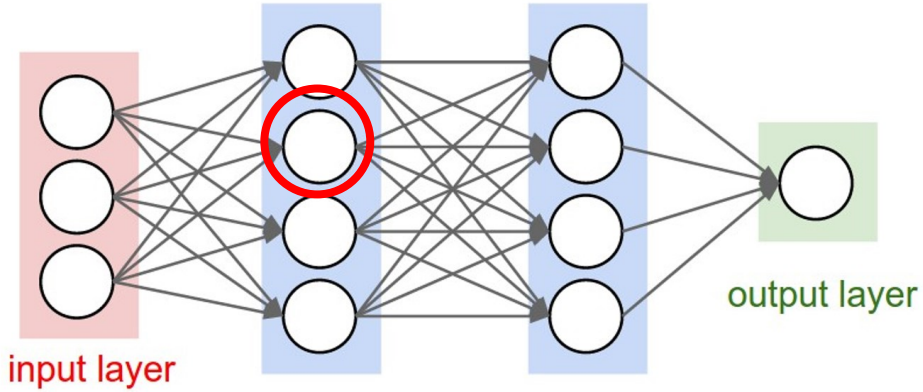


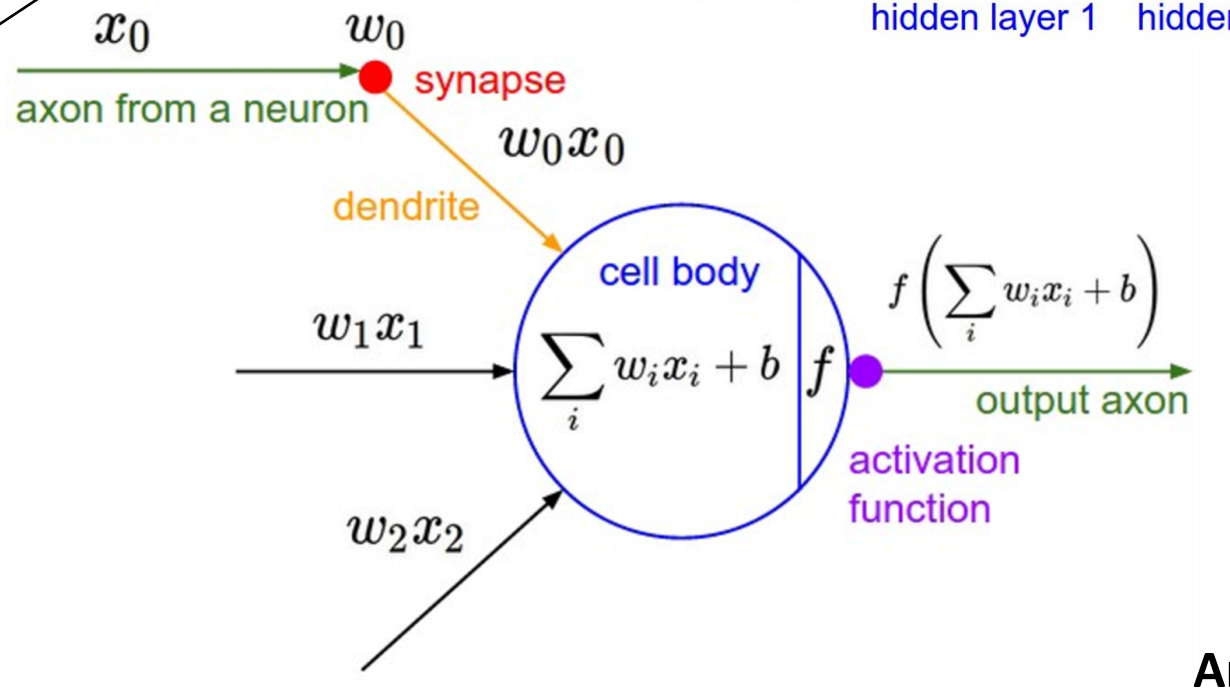
ReLU is generally a good choice for most problems

Biological Neuron vs. Artificial Neuron

Biological Neuron







Artificial Neuron

Problem: **How to Compute Gradients?**

- Nonlinear Score Function:

$$f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$

- Data Loss:

$$\mathcal{L}_i = \sum_{j \neq y_i} \max\left(0, s_j - s_{y_i} + 1\right)$$

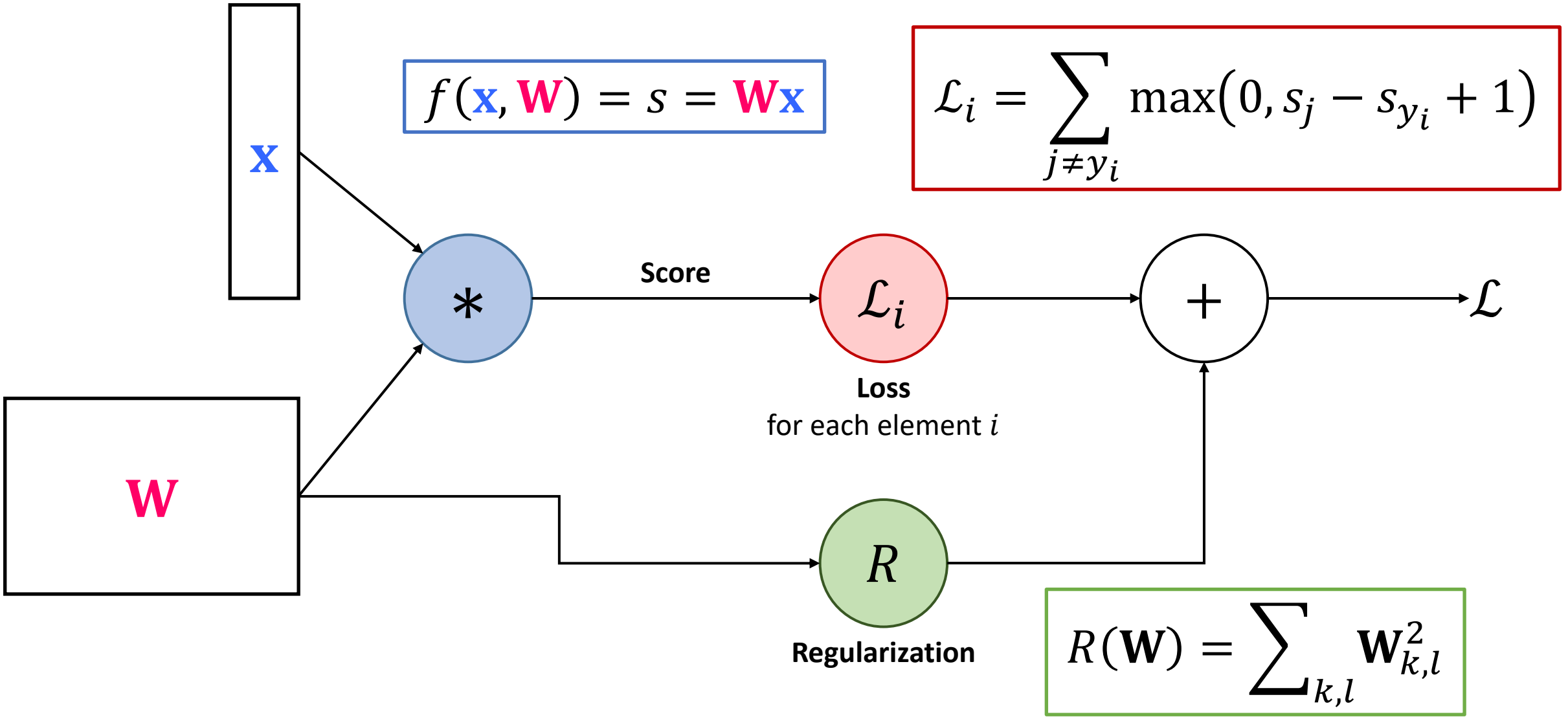
- Regularization:

$$R(\mathbf{W}) = \sum_{k,l} \mathbf{W}_{k,l}^2$$

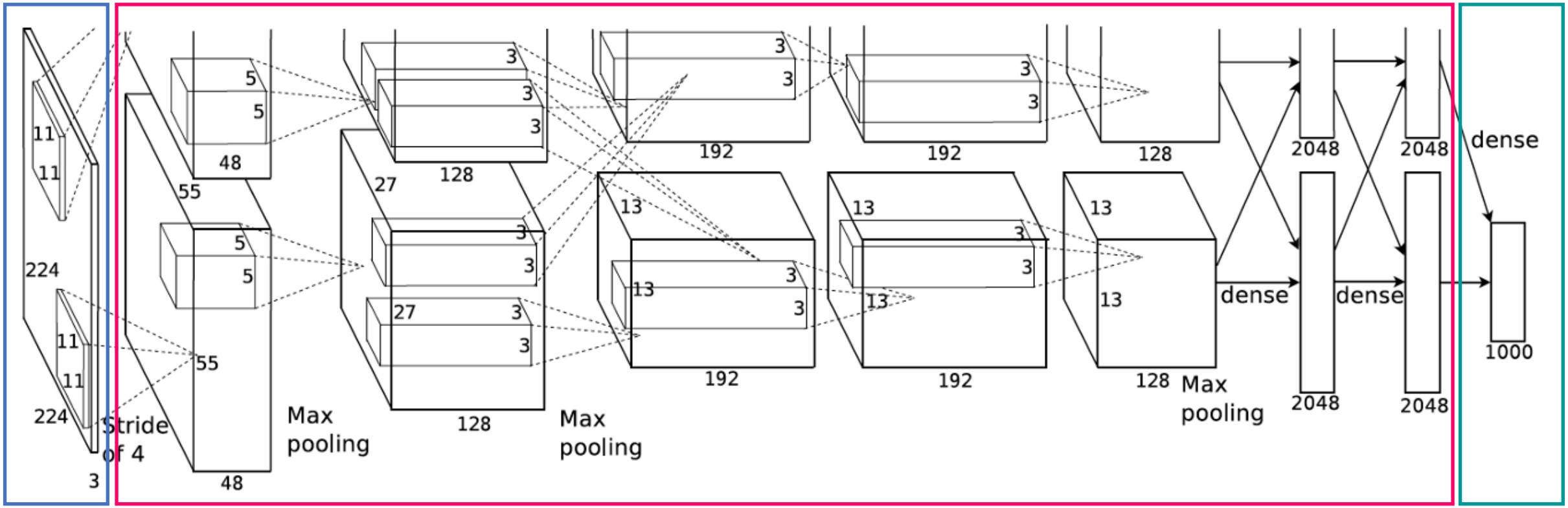
- Total Loss:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i \left(f(\mathbf{x}_i; \mathbf{W}), y_i\right) + \lambda R(\mathbf{W})$$

Idea: **Computational Graphs**



Example of Deep Neural Network: AlexNet



Input image, $\mathbf{x}$

Weights, $\mathbf{W}$

Loss, $\mathcal{L}$

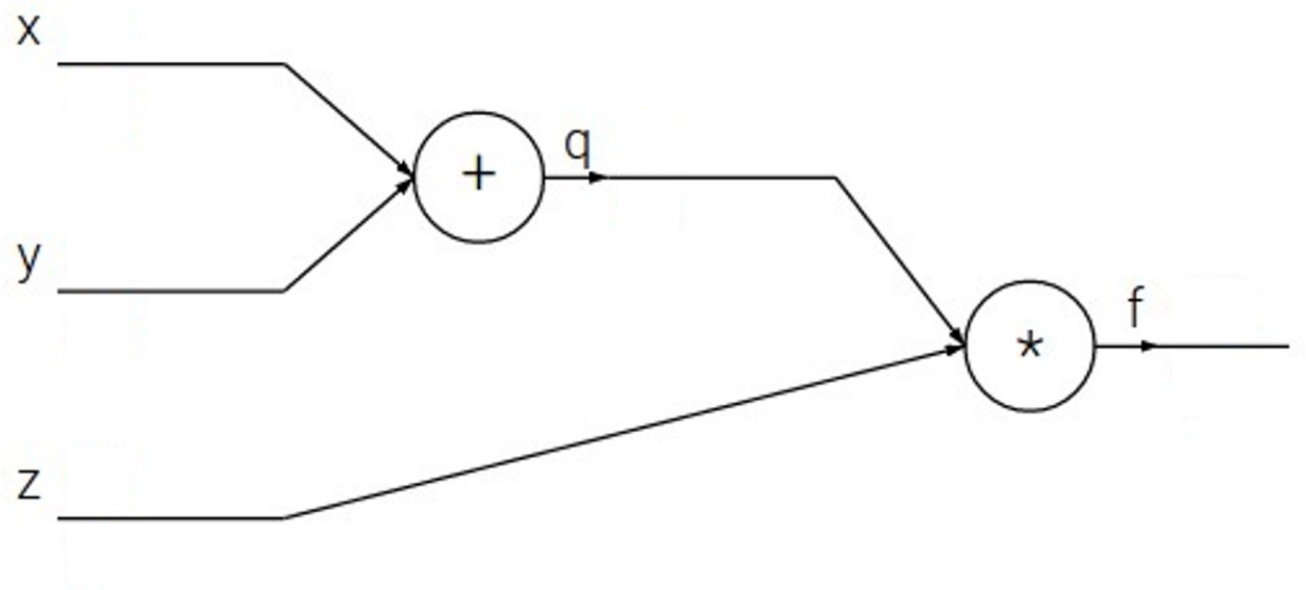
ImageNet Classification with Deep Convolutional Neural Networks, **NeurIPS 2012**

Backpropagation

- In machine learning, backpropagation is a widely used algorithm for **training feedforward neural networks**
- Backpropagation computes **the gradient of the loss function** with respect to the weights of the network for a single input–output example
  - The backpropagation works by computing the gradient of the loss function with respect to each weight **by the chain rule**, computing the gradient one layer at a time, **iterating backward from the last layer** to avoid redundant calculations of intermediate terms in the chain rule
- It does so **efficiently**, unlike a naive direct computation of the gradient with respect to each weight individually

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



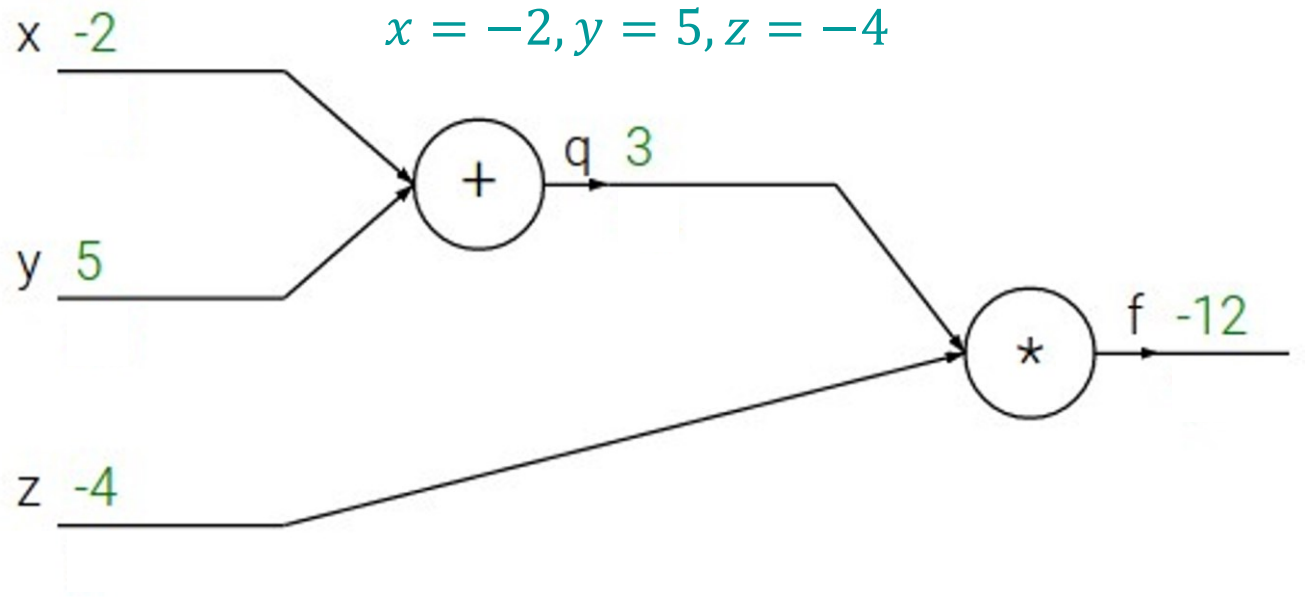
Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

- **Forward Pass:** Compute outputs

– $q = x + y$

– $f = q \cdot z$



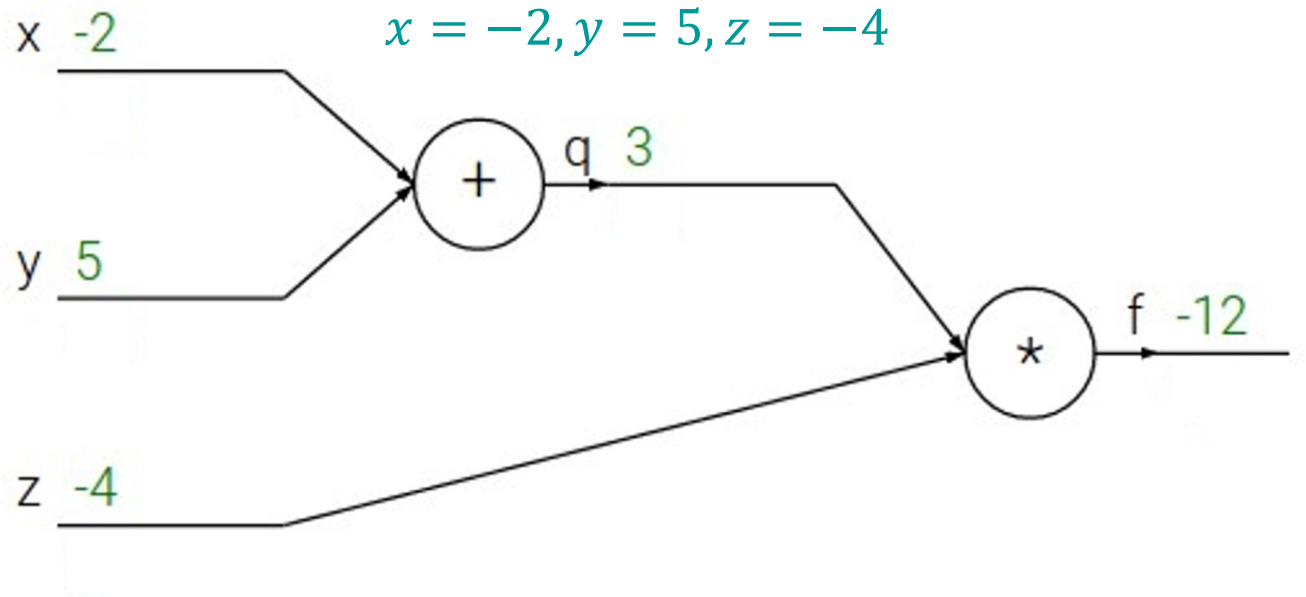
Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

- **Forward Pass:** Compute outputs

– $q = x + y = -2 + 5 = 3$

– $f = q \cdot z = 3 \cdot (-4) = -12$



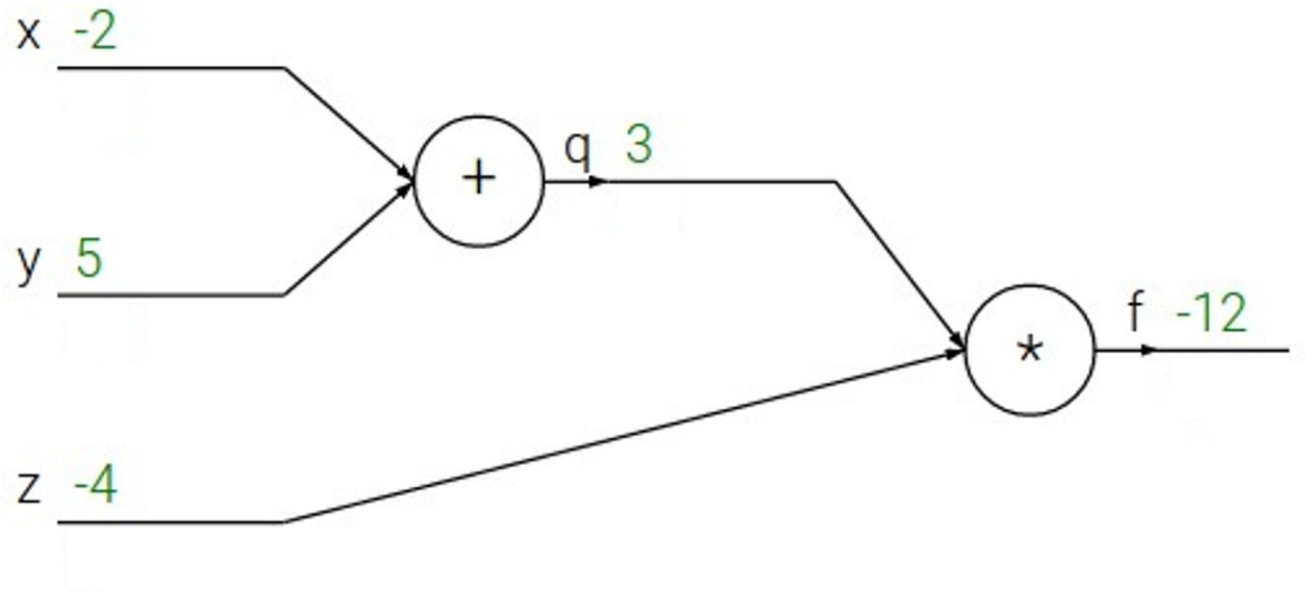
Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

- **Forward Pass:** Compute outputs

– $q = x + y = -2 + 5 = 3$

– $f = q \cdot z = 3 \cdot (-4) = -12$



- **Backward Pass:** Compute derivatives

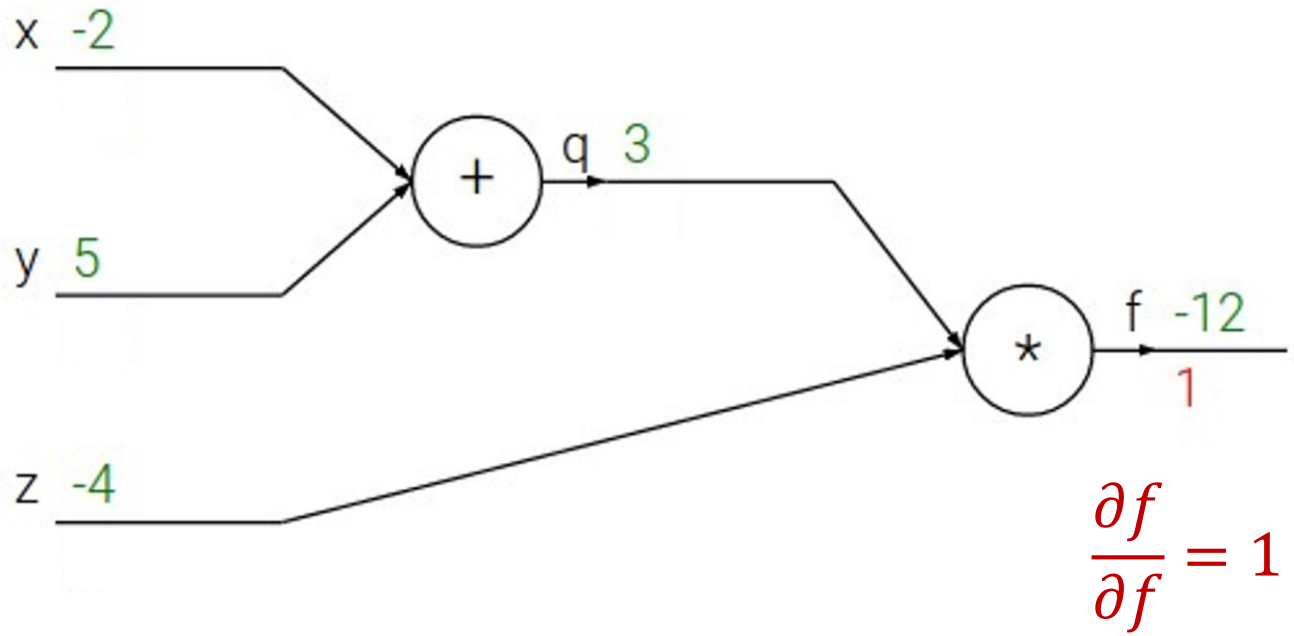
Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

- **Forward Pass:** Compute outputs

– $q = x + y = -2 + 5 = 3$

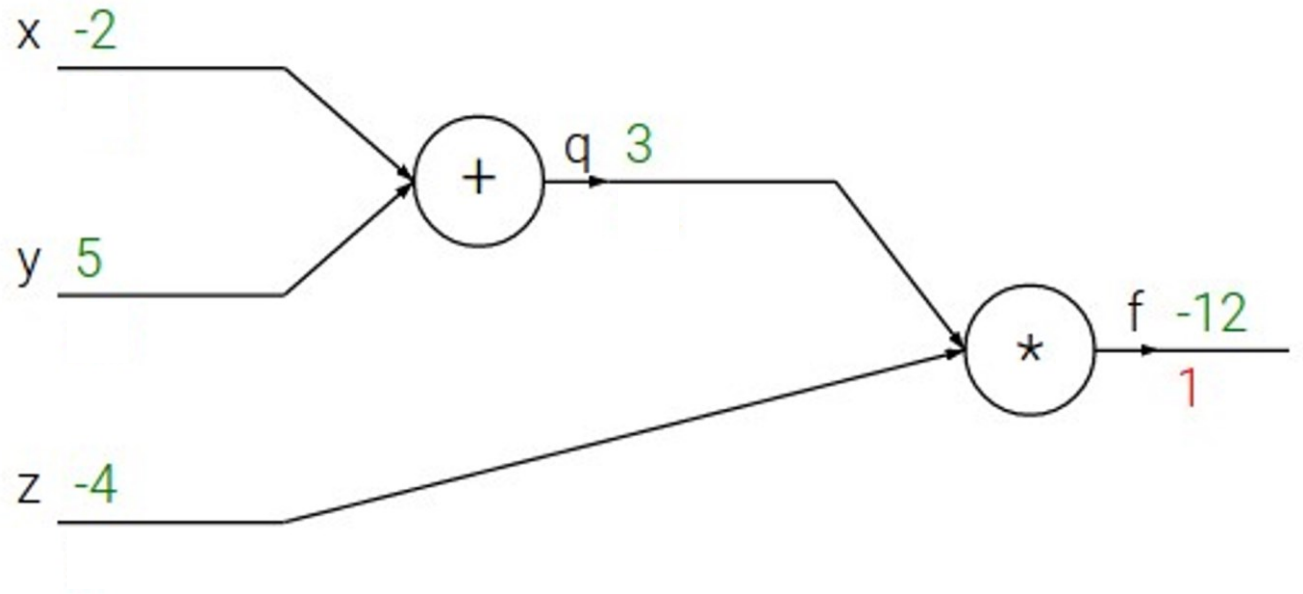
– $f = q \cdot z = 3 \cdot (-4) = -12$



- **Backward Pass:** Compute derivatives

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$

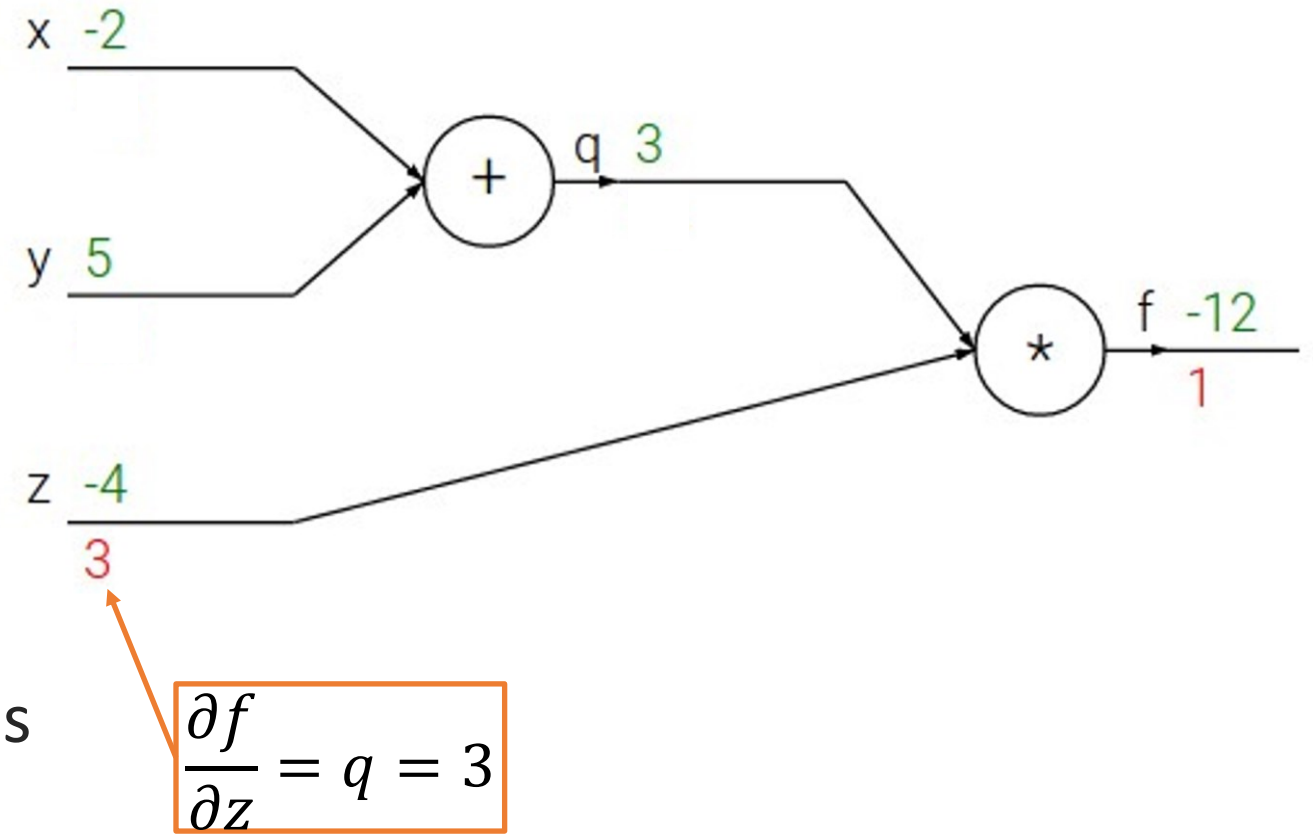
- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

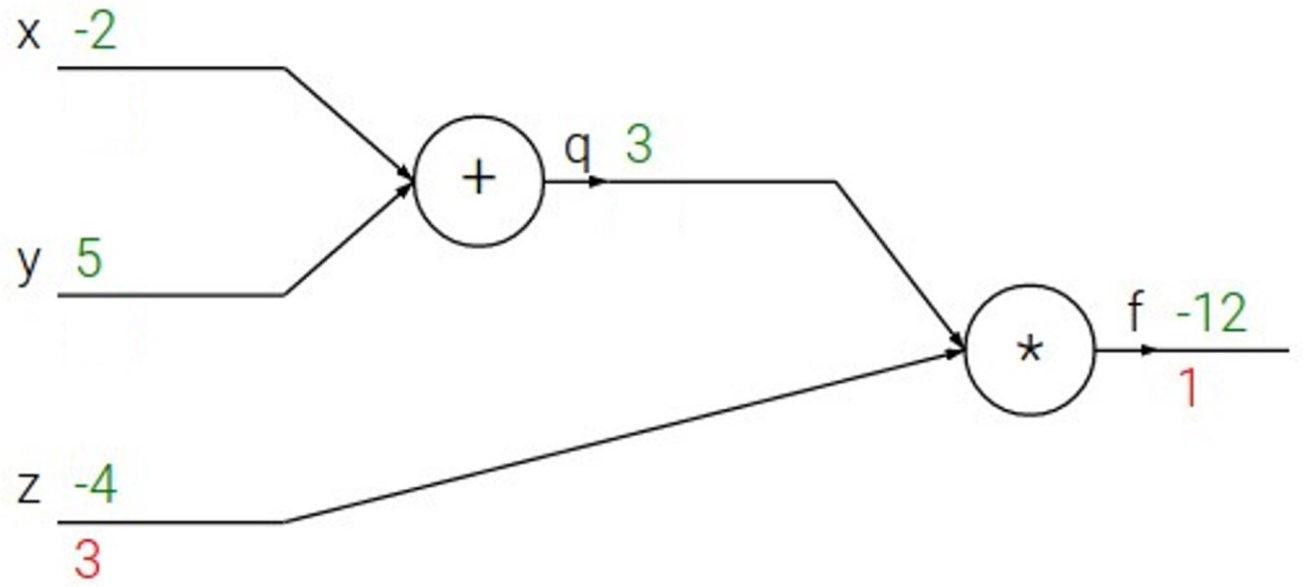
- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$

- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$



Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



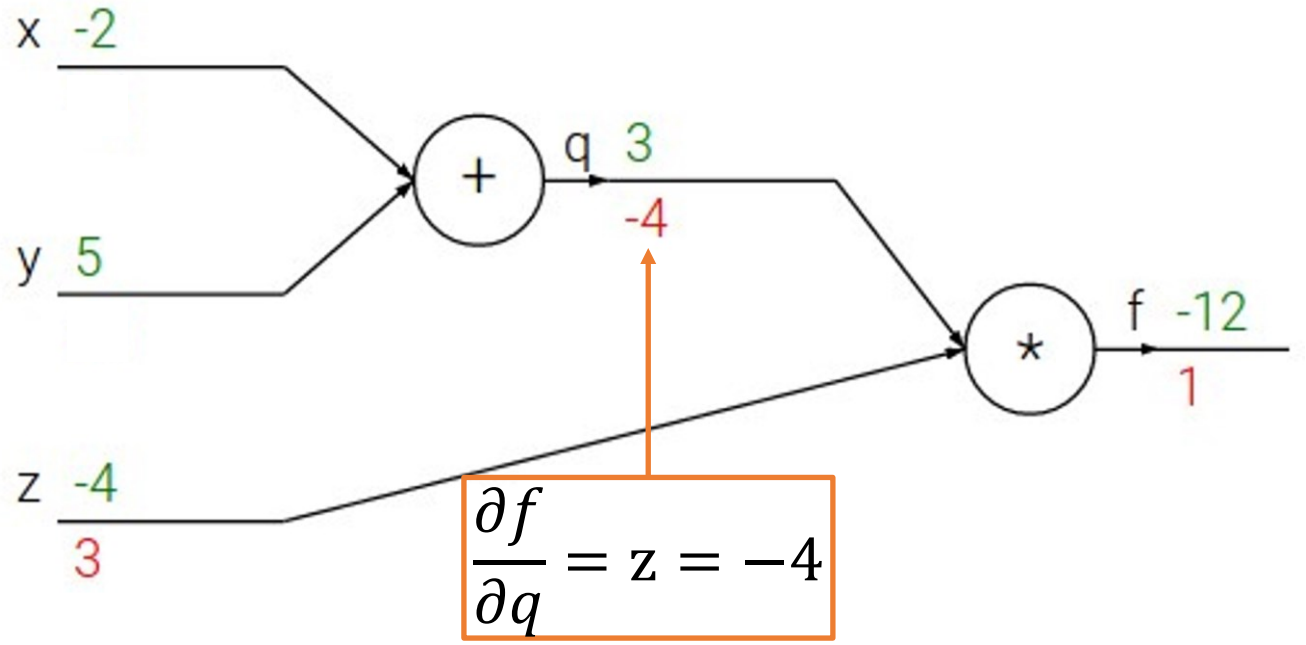
- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$

- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$

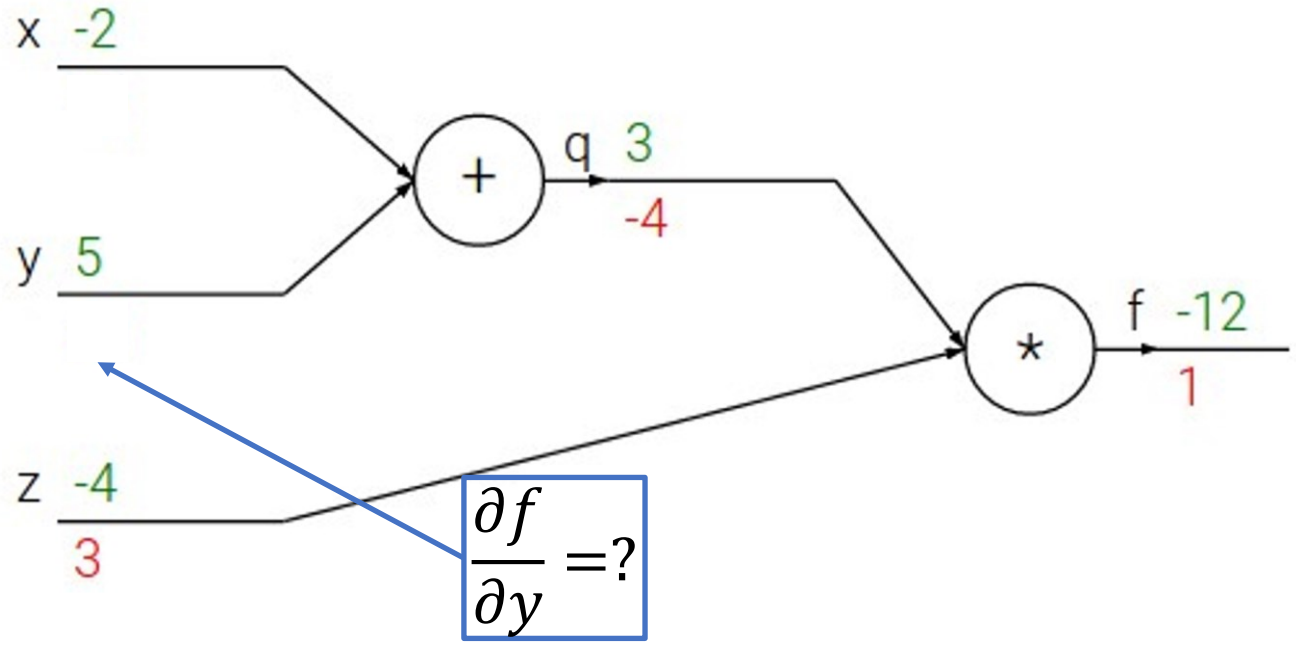


- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$

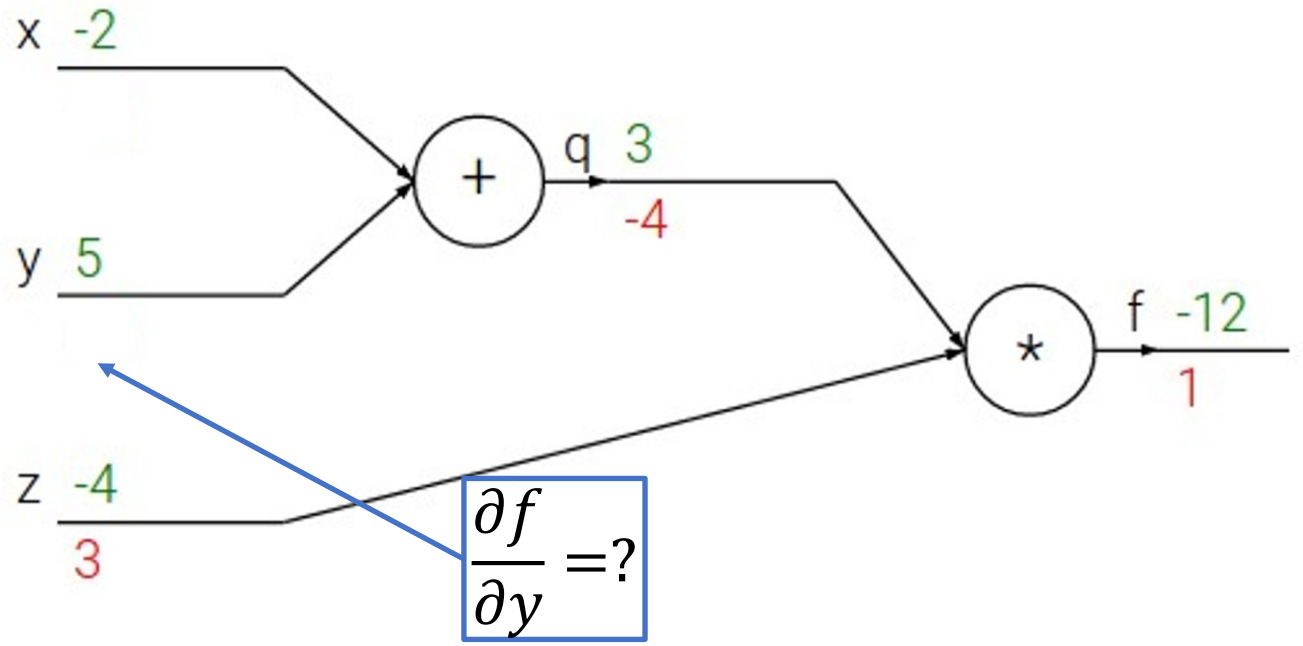
- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$



- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$
  - $\frac{\partial f}{\partial y} = ?$

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$
- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$
  - $\frac{\partial f}{\partial y} = ?$
- **Chain Rule**

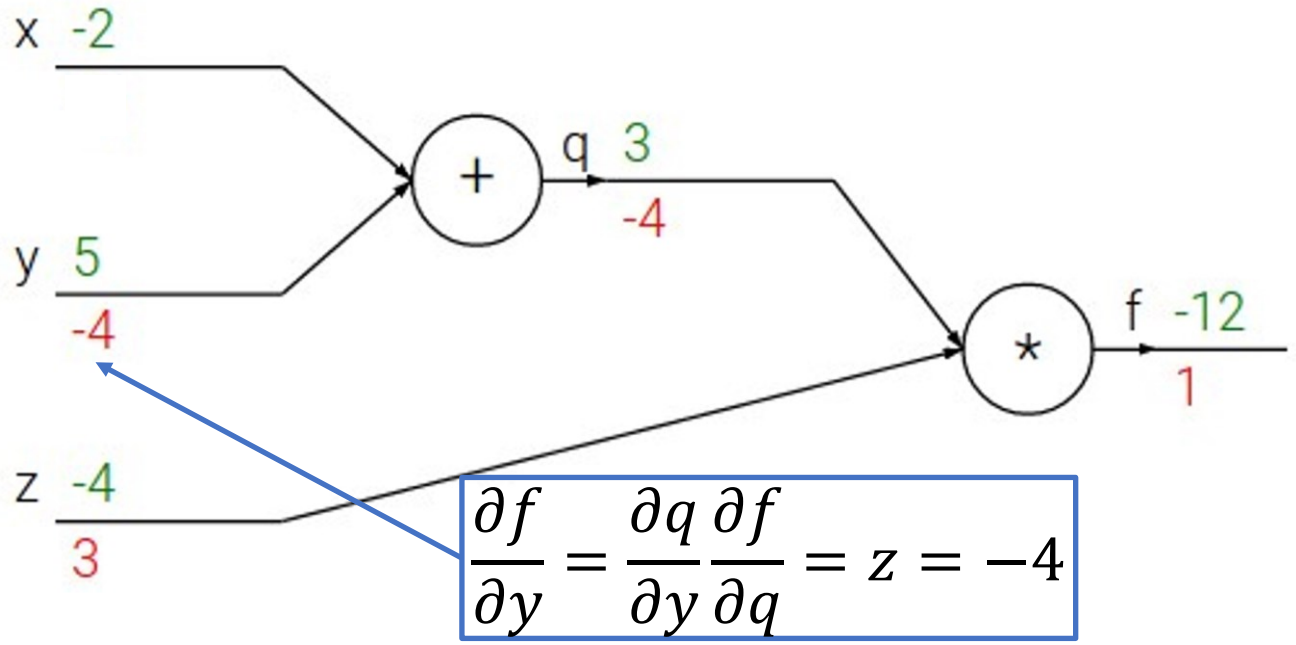
$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

Downstream gradient | Local gradient | Upstream gradient

$$\frac{\partial q}{\partial y} = 1$$

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



- **Forward Pass:** Compute outputs
  - $\boxed{q = x + y} = -2 + 5 = 3$
  - $\boxed{f = q \cdot z} = 3 \cdot (-4) = -12$

- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$
  - $\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q} = z}$

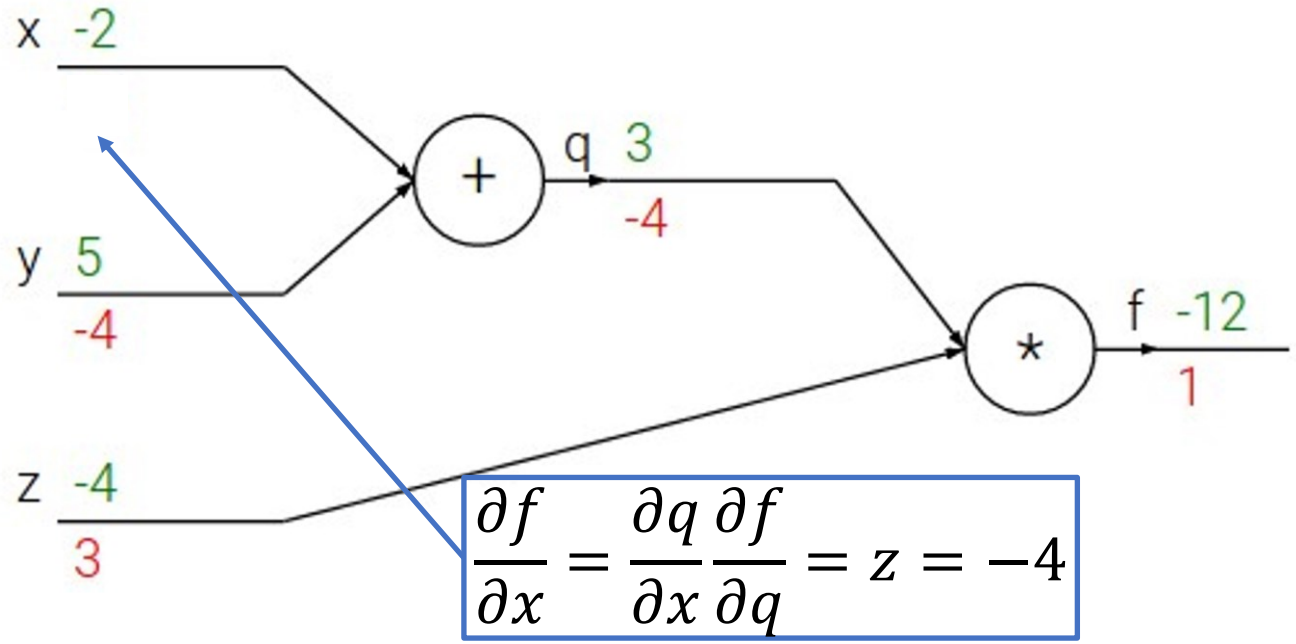
- **Chain Rule**

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

Downstream gradient | Local gradient | Upstream gradient

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$

- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$
  - $\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q} = z$
  - $\frac{\partial f}{\partial x} = ?$

- **Chain Rule**

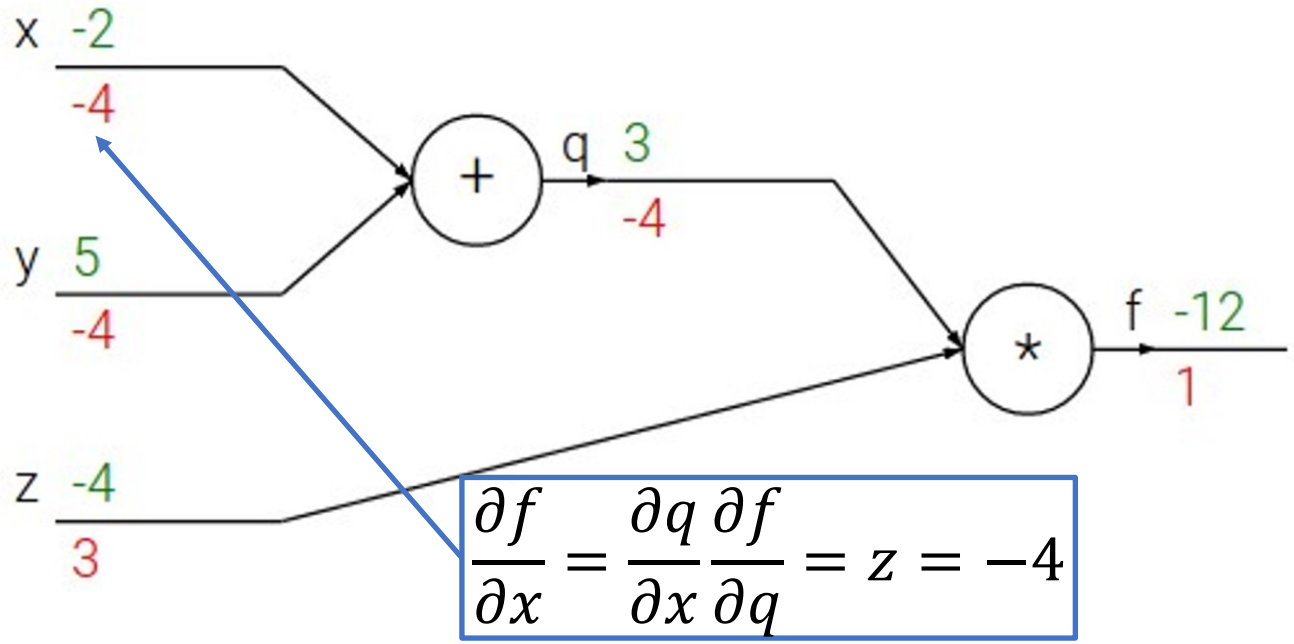
$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

Downstream gradient | Local gradient | Upstream gradient

$$\frac{\partial q}{\partial x} = 1$$

Example I: Backpropagation

- $f(x, y, z) = (x + y) \cdot z$



- **Forward Pass:** Compute outputs
  - $q = x + y = -2 + 5 = 3$
  - $f = q \cdot z = 3 \cdot (-4) = -12$

- **Backward Pass:** Compute derivatives
  - $\frac{\partial f}{\partial z} = q$
  - $\frac{\partial f}{\partial q} = z$
  - $\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q} = z$
  - $\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q} = z$

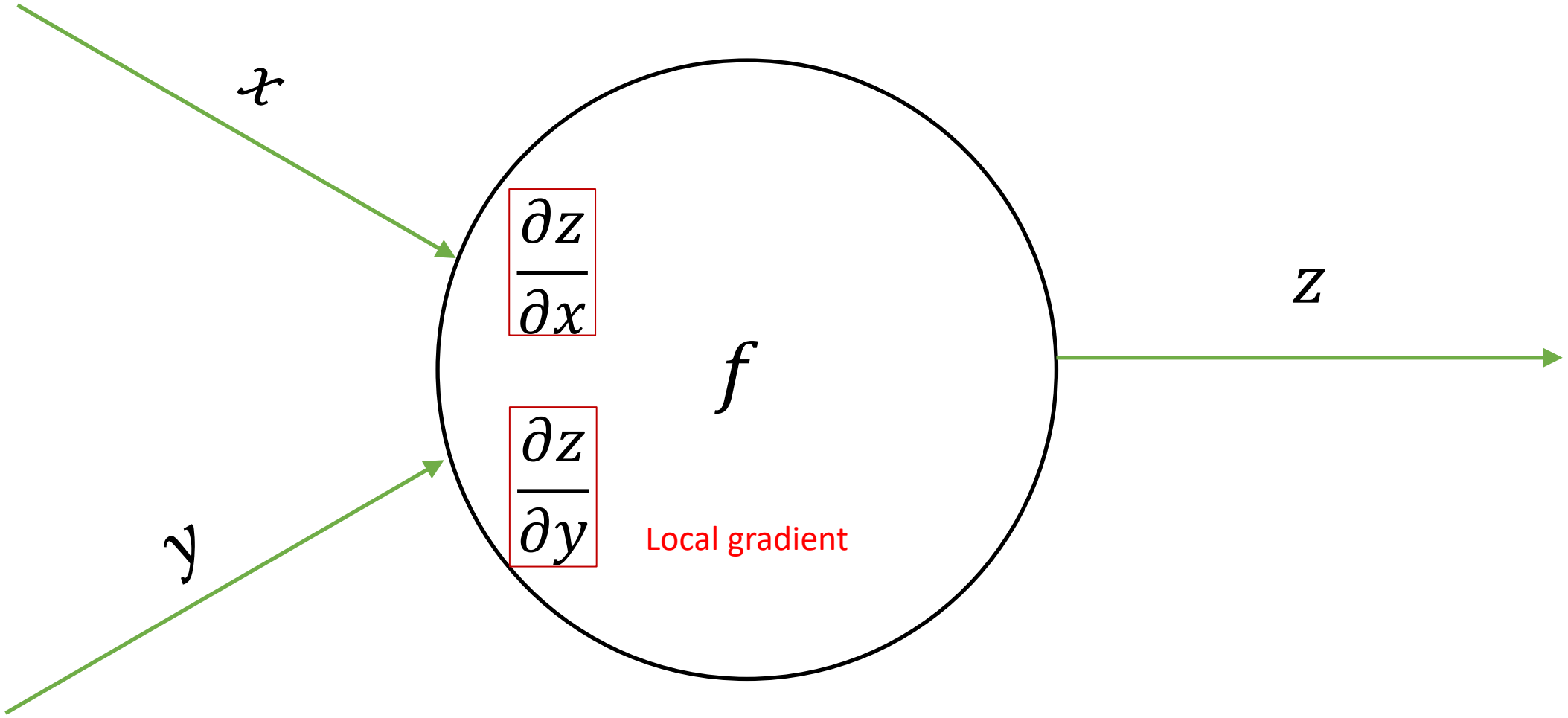
- **Chain Rule**

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

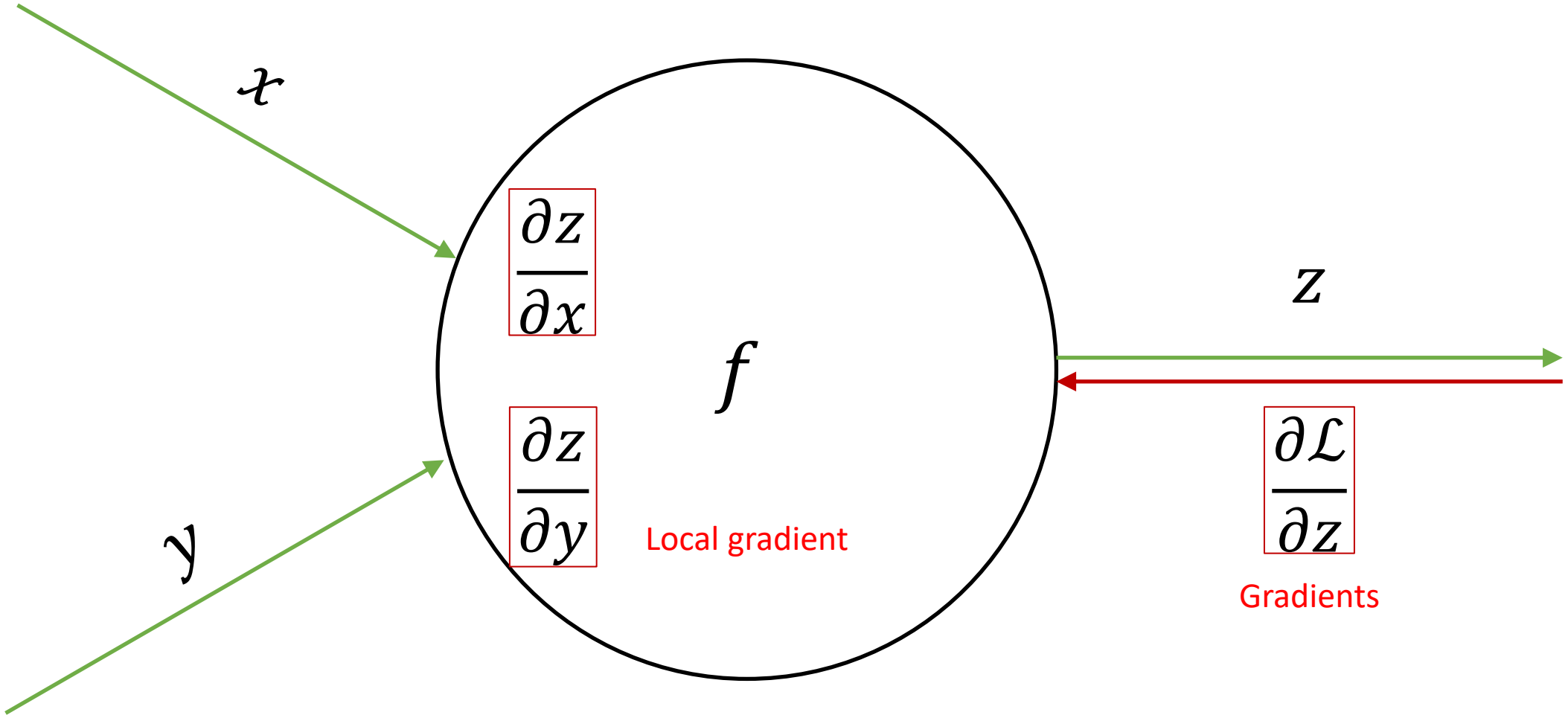
Downstream gradient / Local gradient / Upstream gradient

$$\frac{\partial q}{\partial x} = 1$$

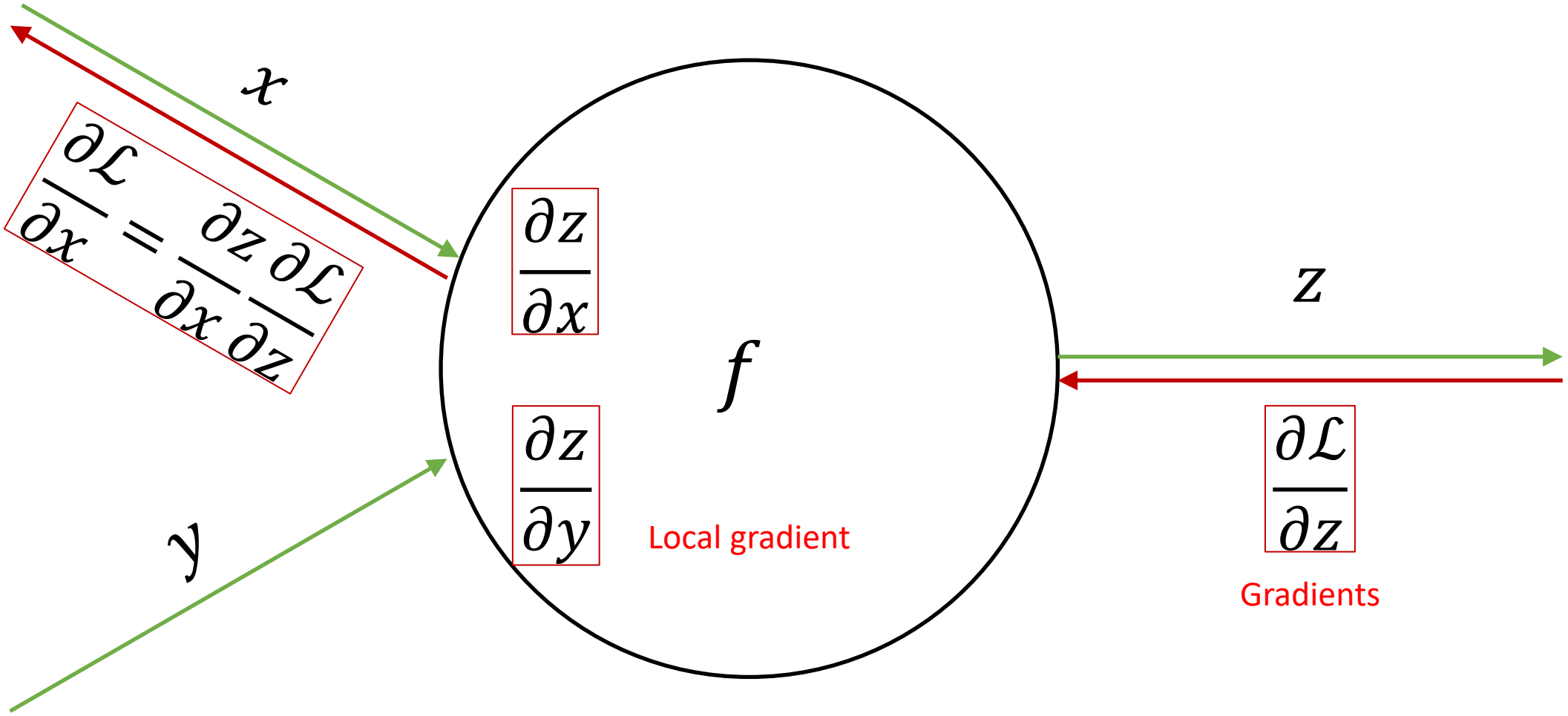
Graphical Example



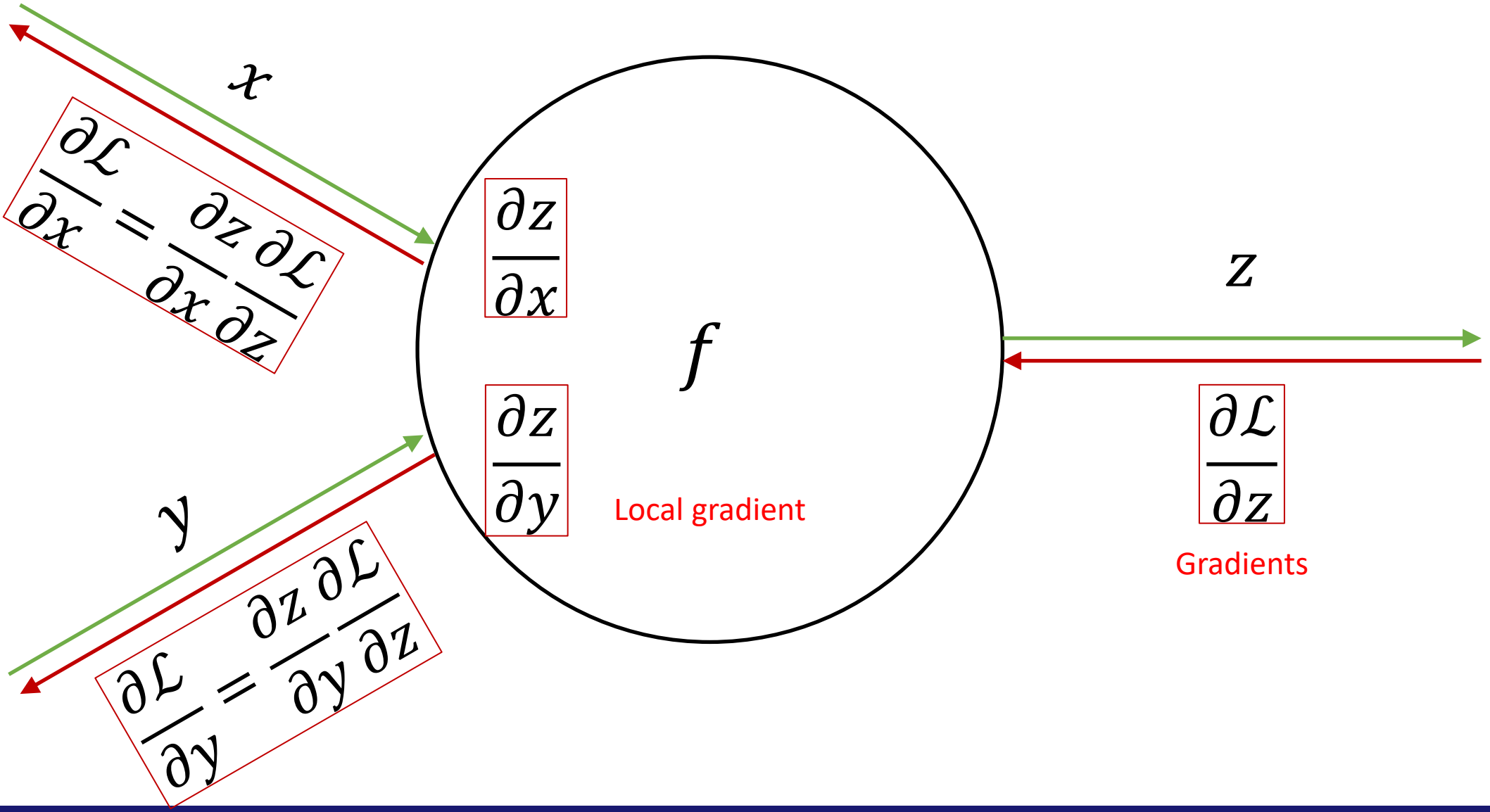
Graphical Example



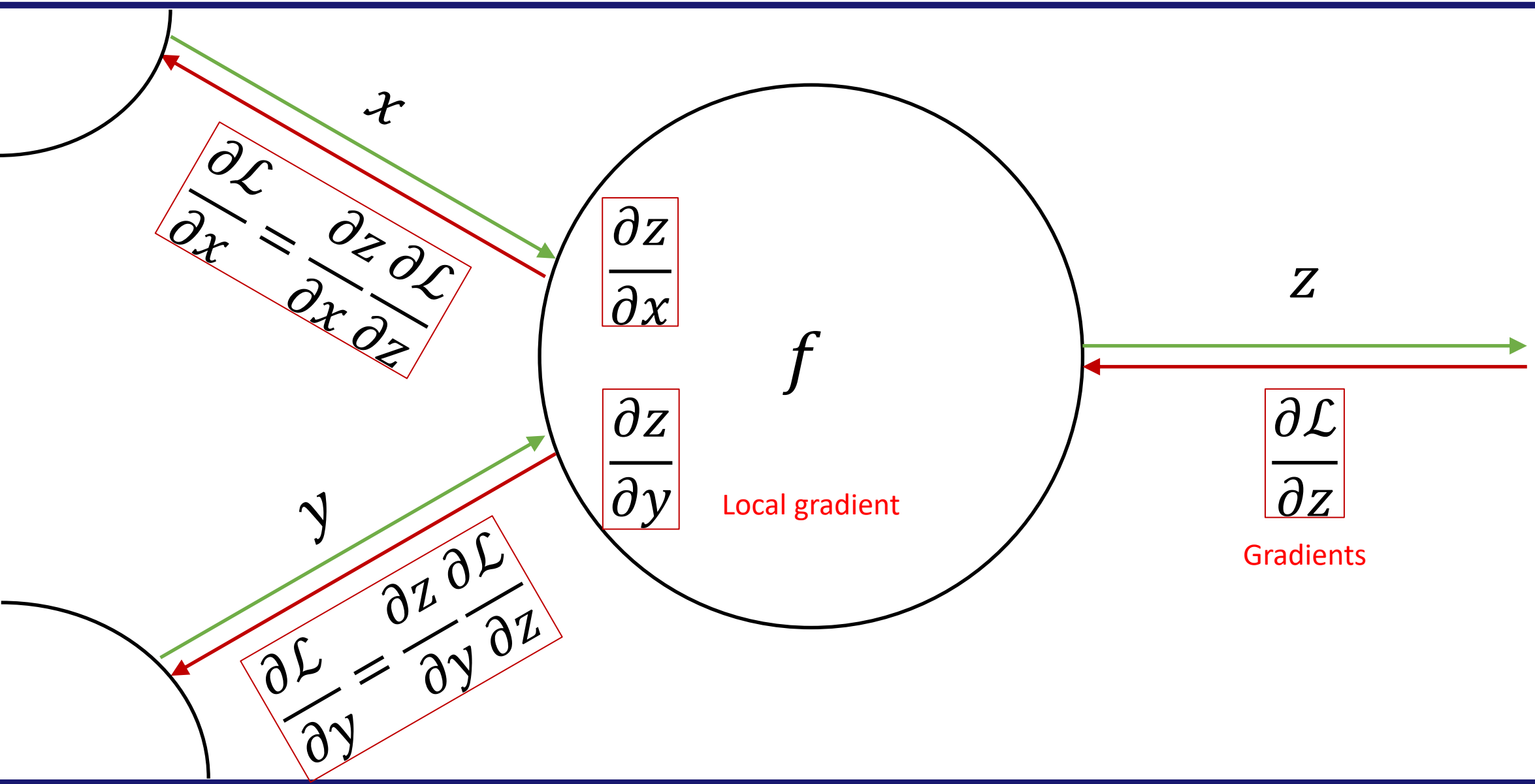
Graphical Example



Graphical Example

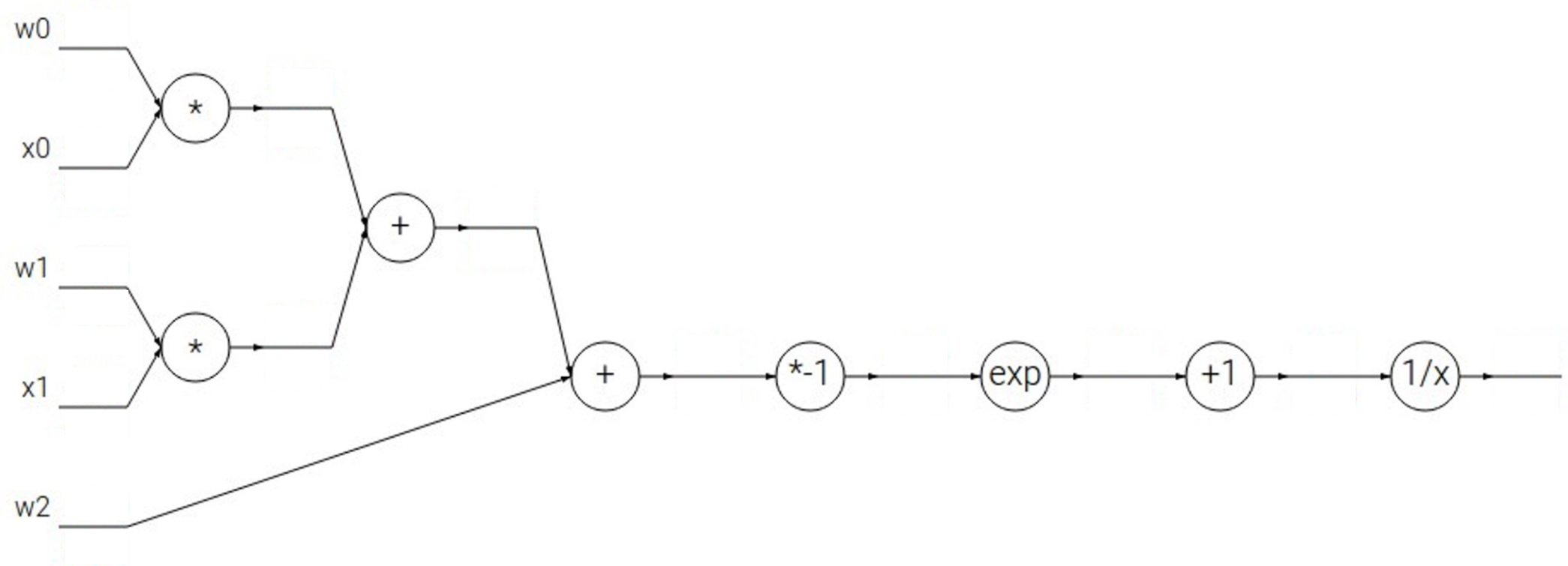


Graphical Example



Example II: Backpropagation

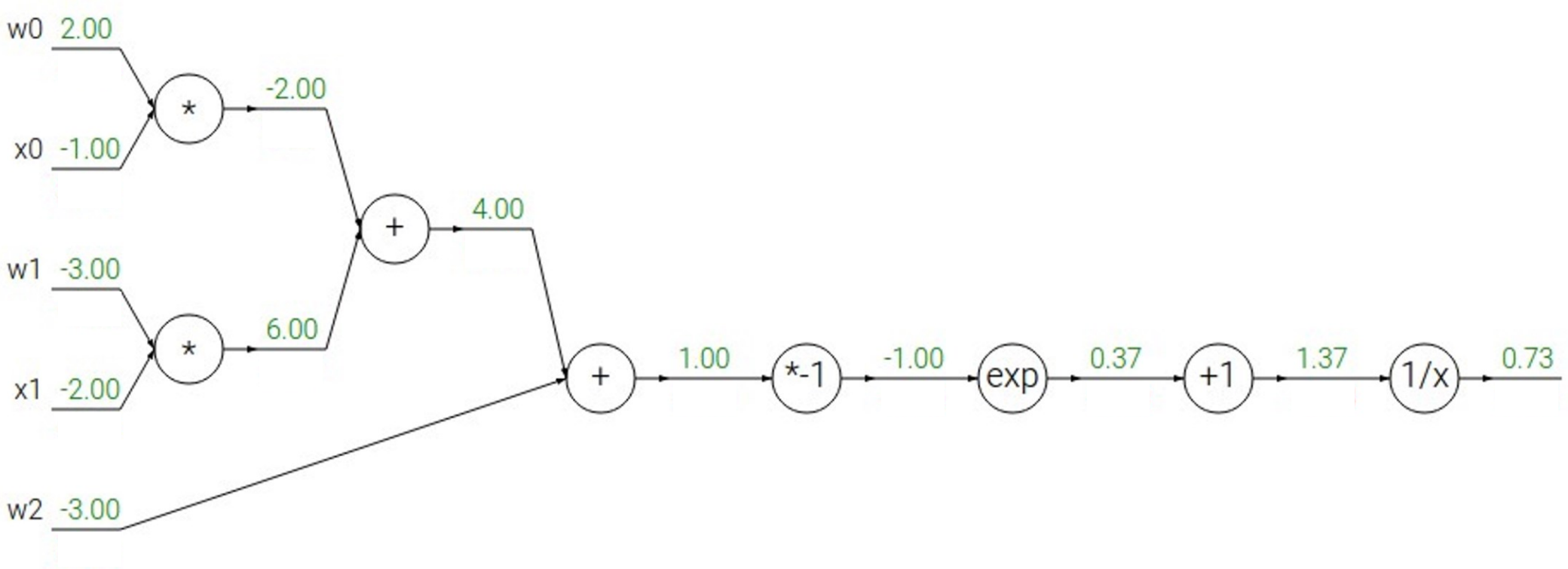
- $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



Example II: Backpropagation

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

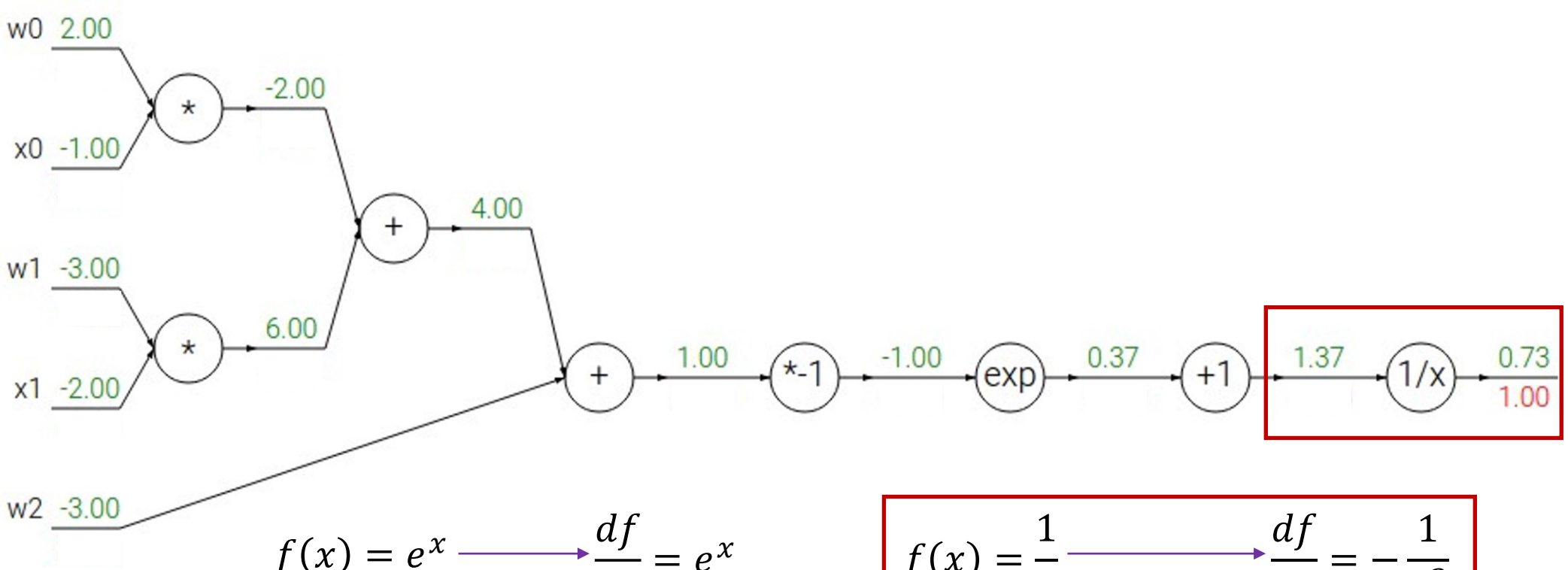
Forward pass: Compute outputs



Example II: Backpropagation

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

Backward pass: Compute gradients



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

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

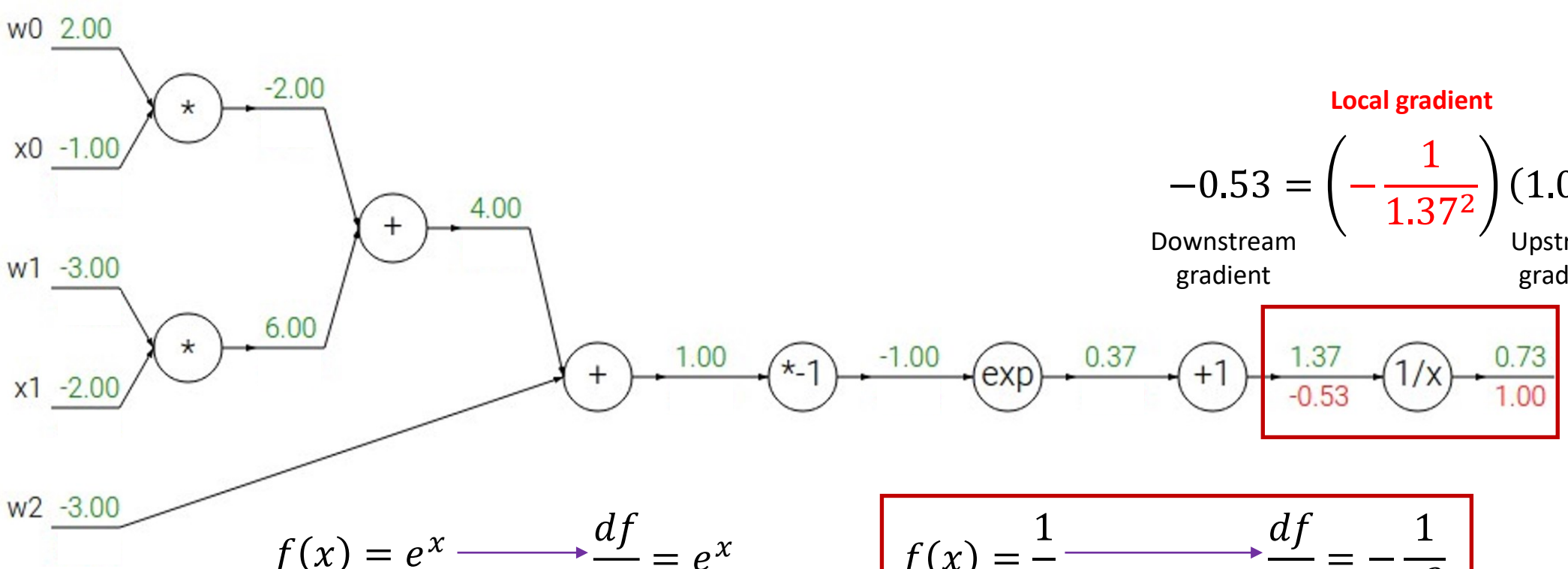
$$f(x) = ax \longrightarrow \frac{df}{dx} = a$$

$$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

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

Downstream gradient

Upstream gradient

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

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

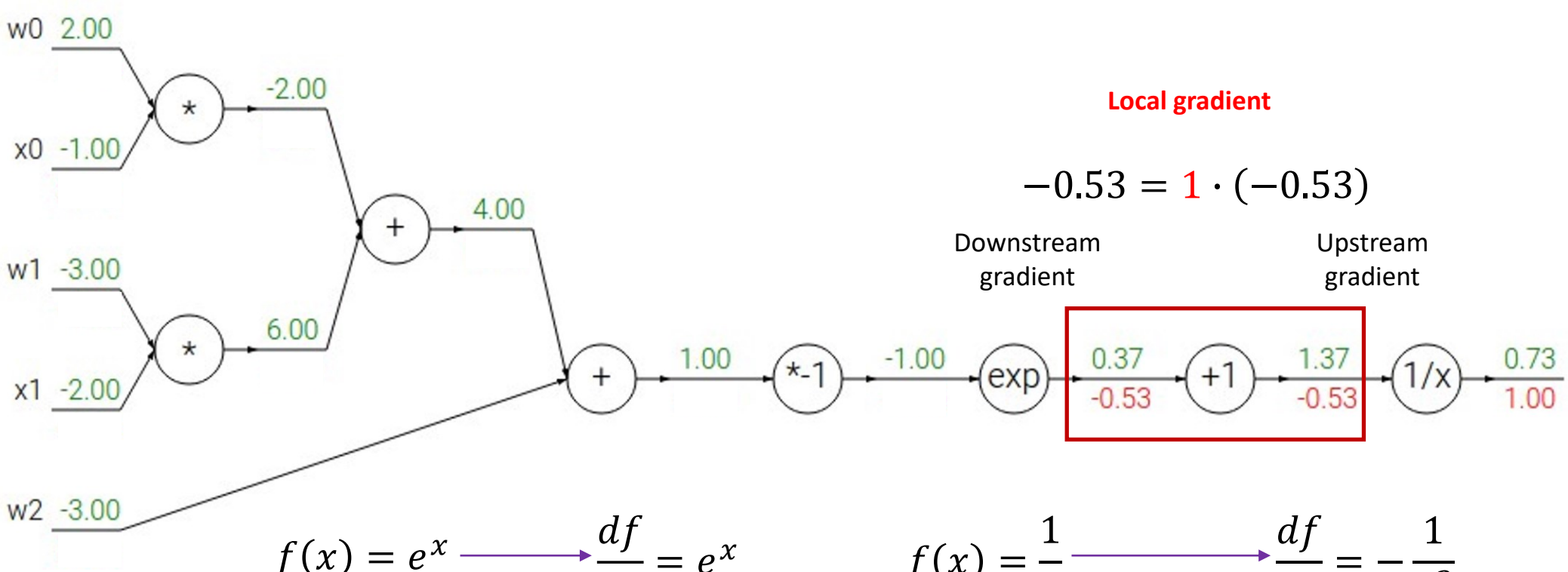
$f(x) = ax \longrightarrow \frac{df}{dx} = a$

$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$

Example II: Backpropagation

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

Backward pass: Compute gradients



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

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

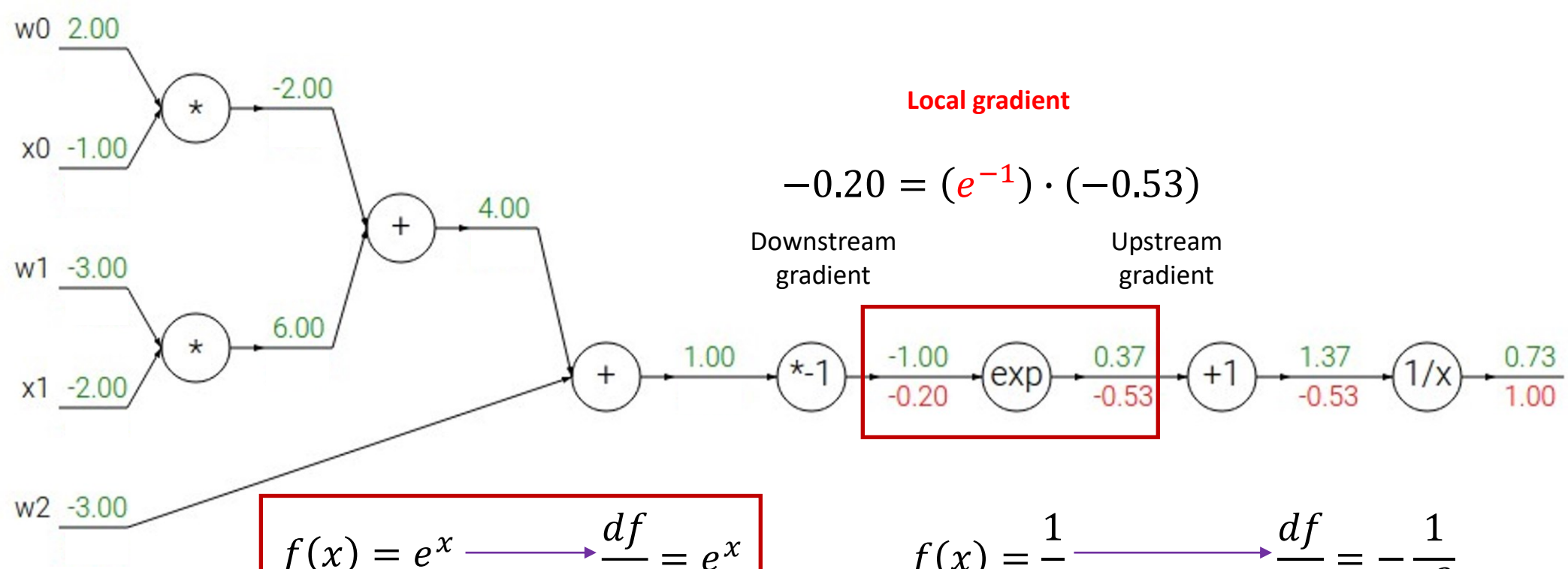
$$f(x) = ax \longrightarrow \frac{df}{dx} = a$$

$$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

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

Downstream gradient

Upstream gradient

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

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

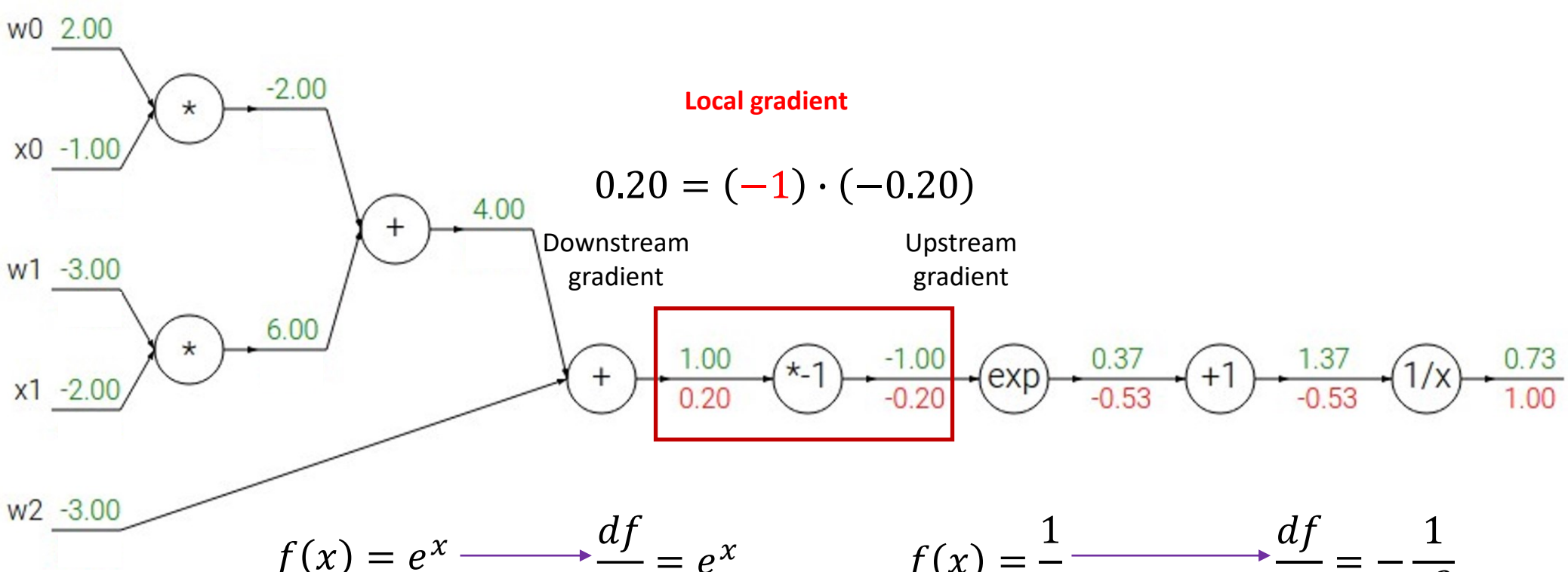
$$f(x) = ax \longrightarrow \frac{df}{dx} = a$$

$$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

$$0.20 = (-1) \cdot (-0.20)$$

Downstream gradient

Upstream gradient

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

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

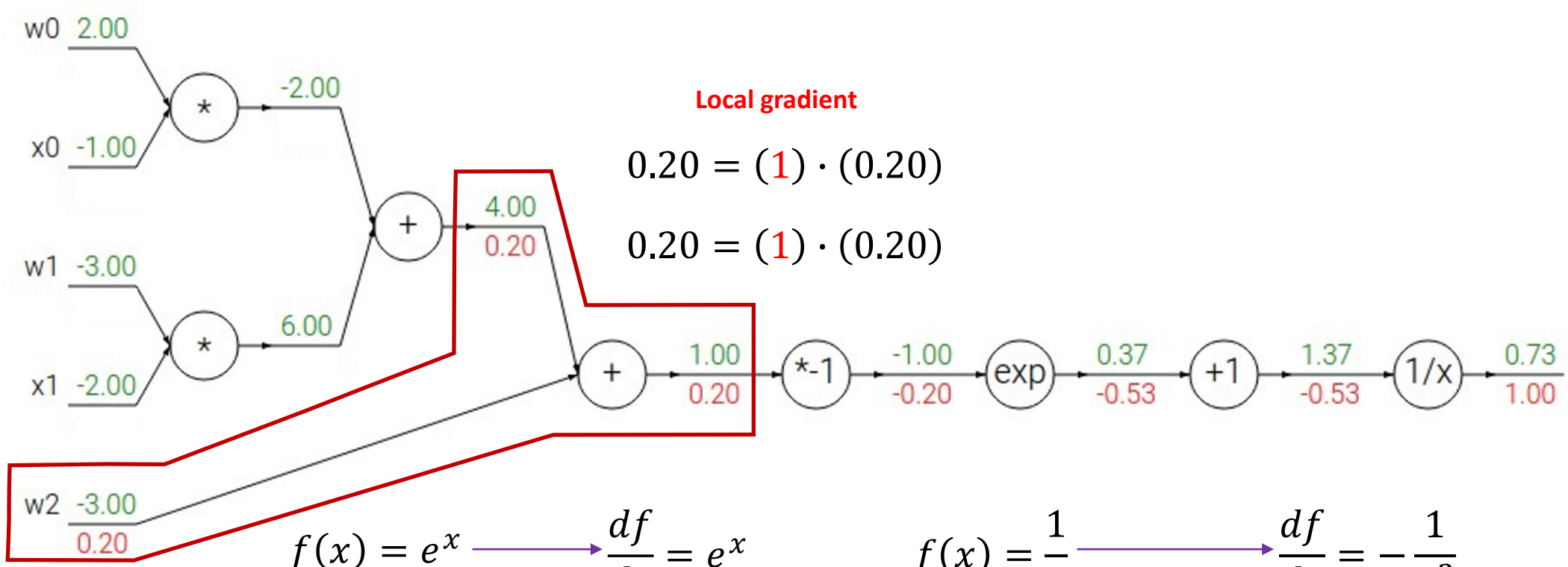
$$f(x) = ax \longrightarrow \frac{df}{dx} = a$$

$$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

$0.20 = (1) \cdot (0.20)$

$0.20 = (1) \cdot (0.20)$

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

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

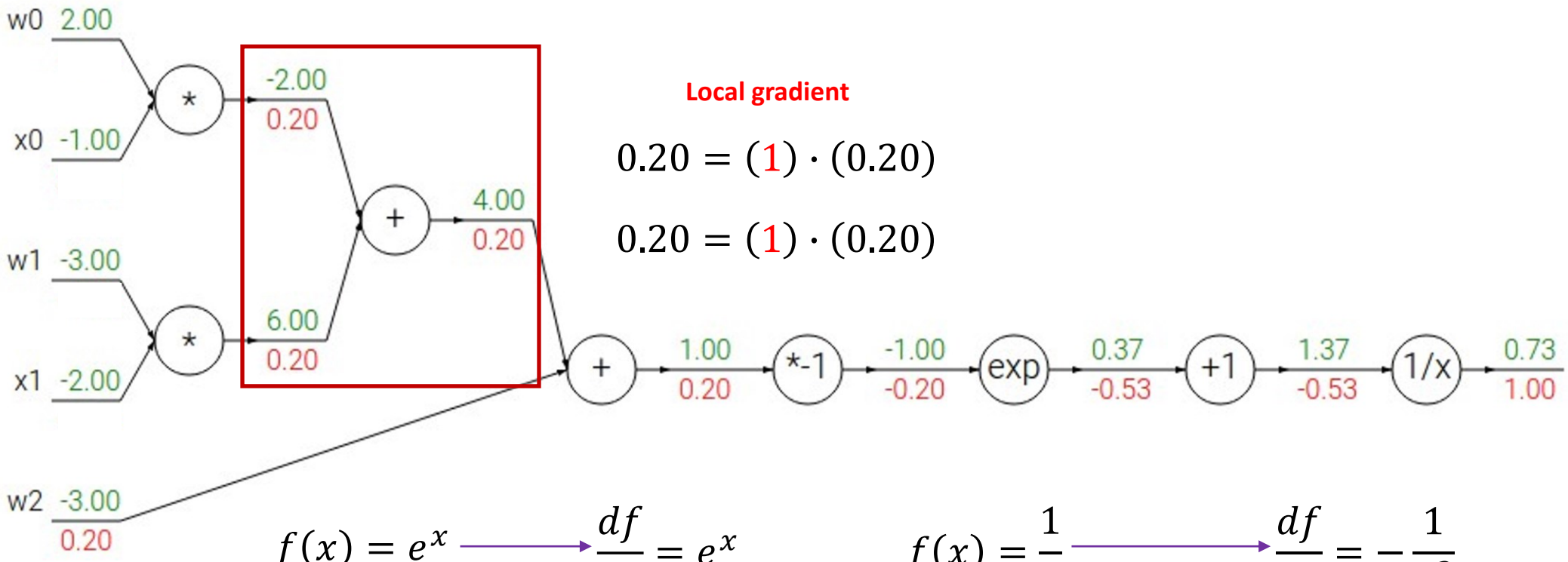
$f(x) = ax \longrightarrow \frac{df}{dx} = a$

$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

$$0.20 = (1) \cdot (0.20)$$

$$0.20 = (1) \cdot (0.20)$$

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

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

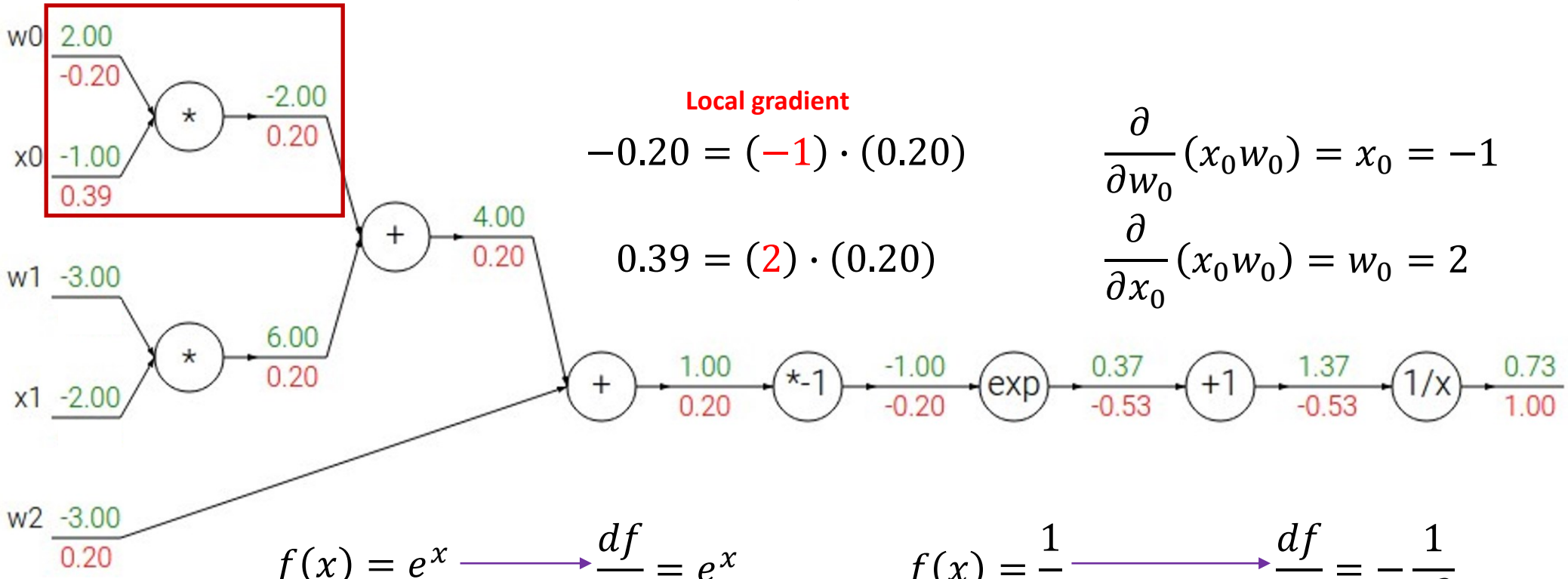
$$f(x) = ax \longrightarrow \frac{df}{dx} = a$$

$$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

$-0.20 = (-1) \cdot (0.20)$

$0.39 = (2) \cdot (0.20)$

$\frac{\partial}{\partial w_0}(x_0w_0) = x_0 = -1$

$\frac{\partial}{\partial x_0}(x_0w_0) = w_0 = 2$

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

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

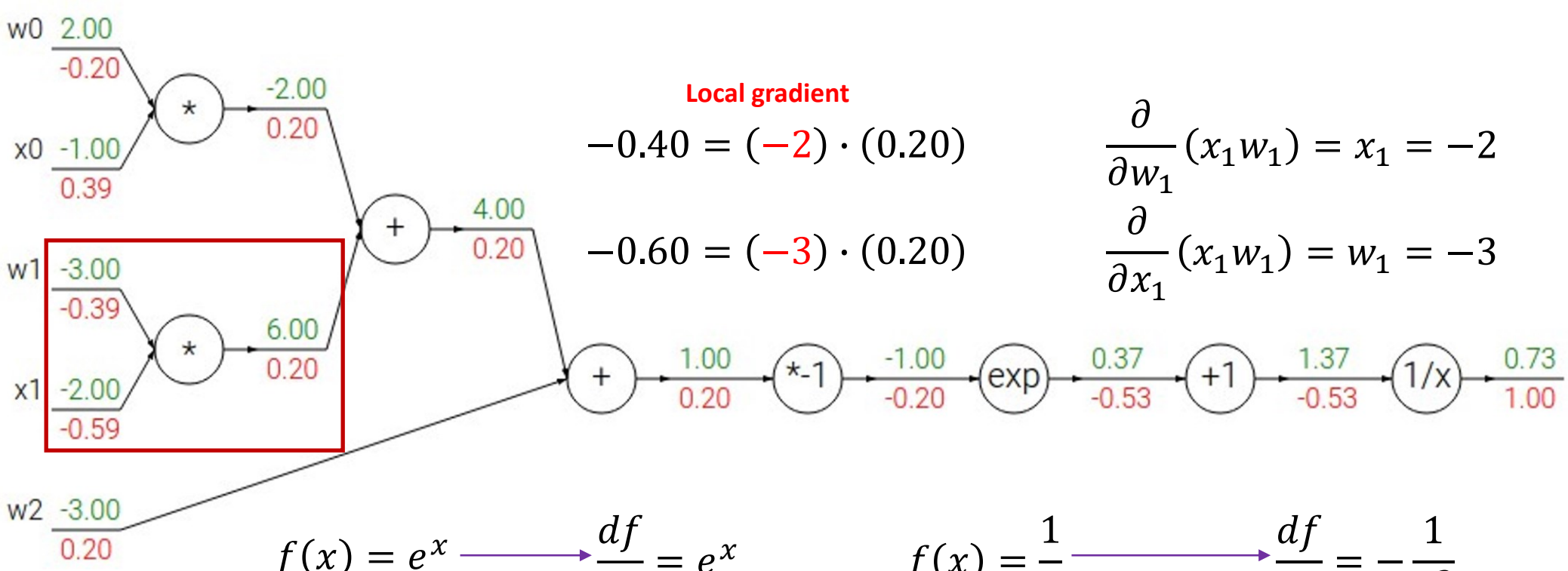
$f(x) = ax \longrightarrow \frac{df}{dx} = a$

$f(x) = x + c \longrightarrow \frac{df}{dx} = 1$

Example II: Backpropagation

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

Backward pass: Compute gradients



Local gradient

$-0.40 = (-2) \cdot (0.20)$

$-0.60 = (-3) \cdot (0.20)$

$\frac{\partial}{\partial w_1}(x_1 w_1) = x_1 = -2$

$\frac{\partial}{\partial x_1}(x_1 w_1) = w_1 = -3$

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

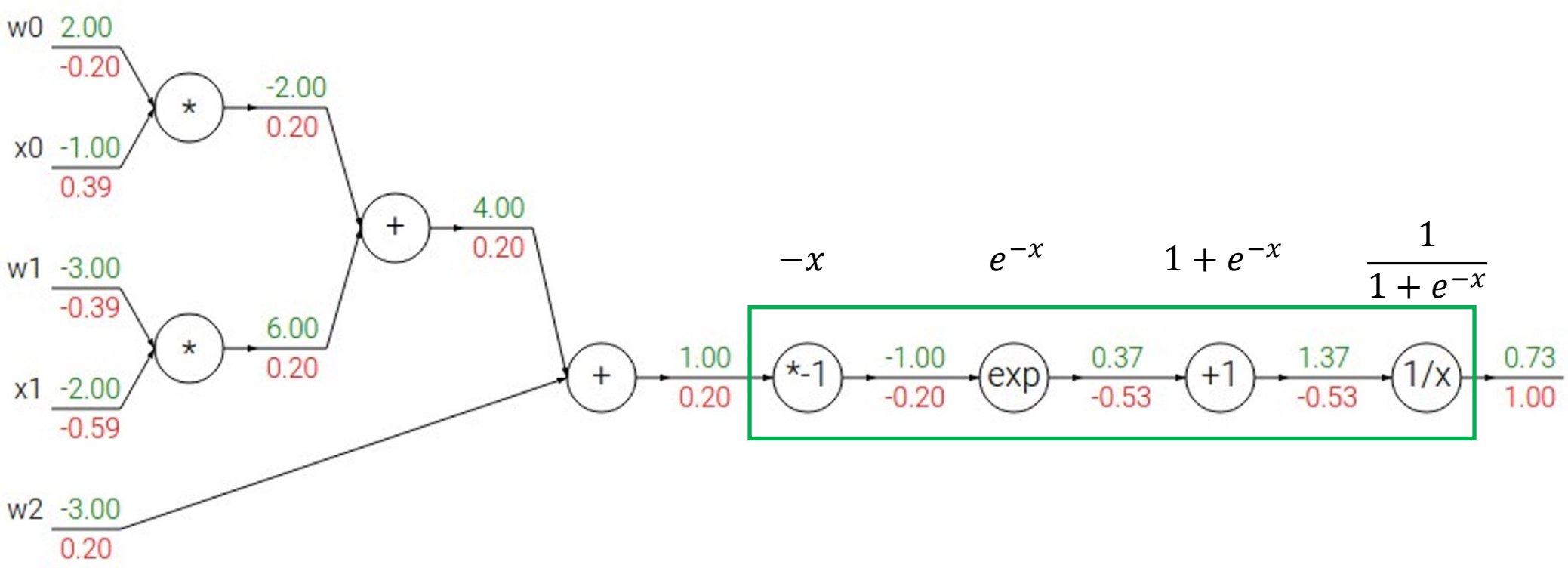
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -\frac{1}{x^2}$

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

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

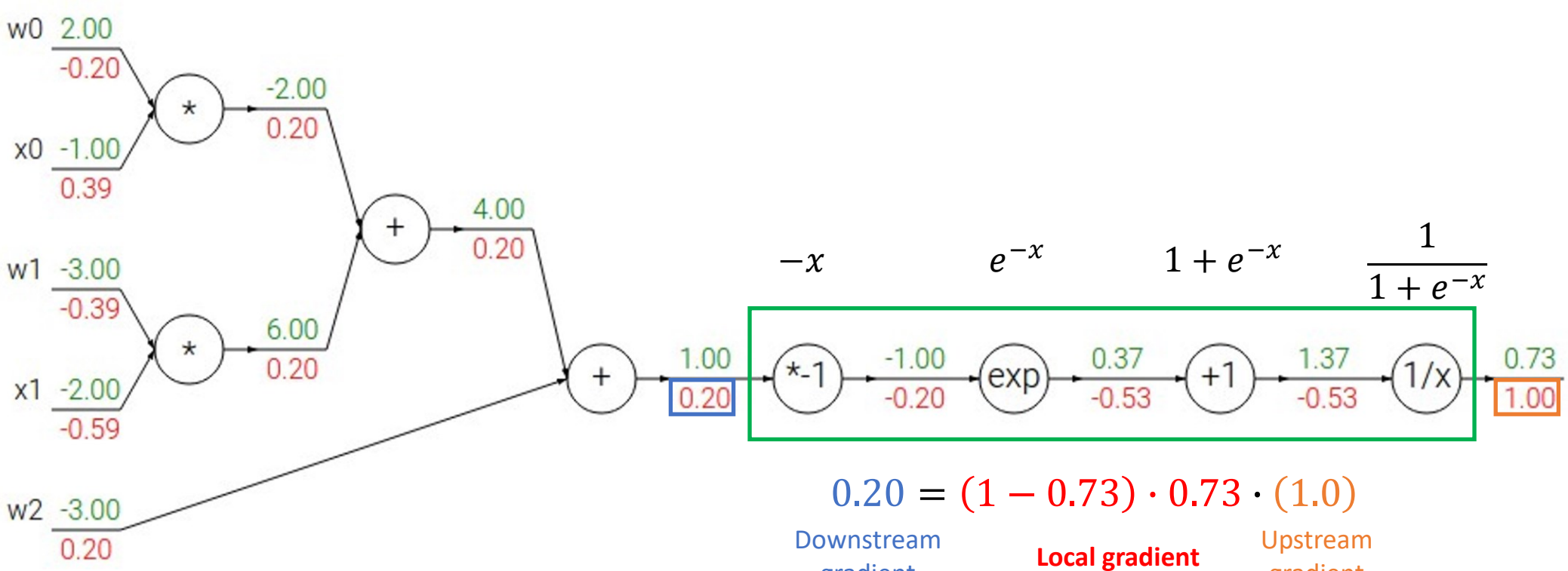
Example II: Backpropagation

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



Example II: Backpropagation

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



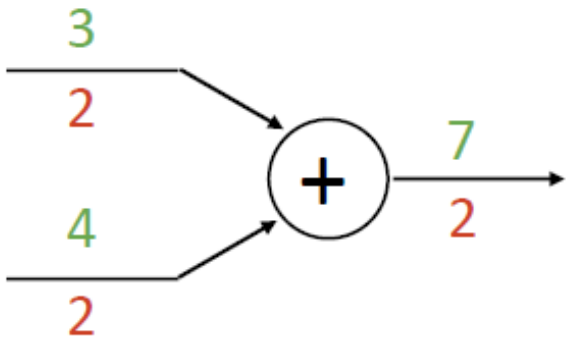
$$0.20 = (1 - 0.73) \cdot 0.73 \cdot (1.0)$$

Downstream gradient | **Local gradient** | Upstream gradient

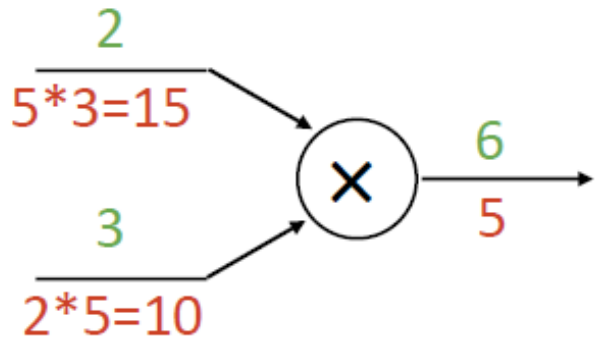
$$\frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = \big(1-\sigma(x)\big)\sigma(x)$$

Patterns in Gradient Flow

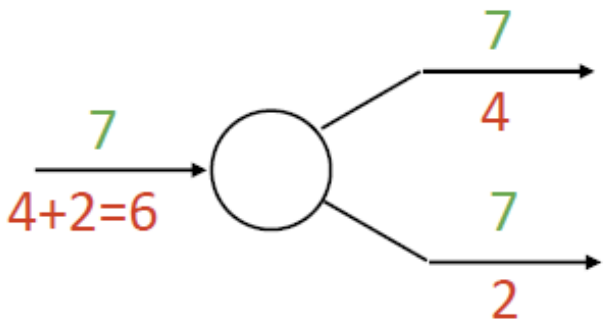
add gate: gradient distributor



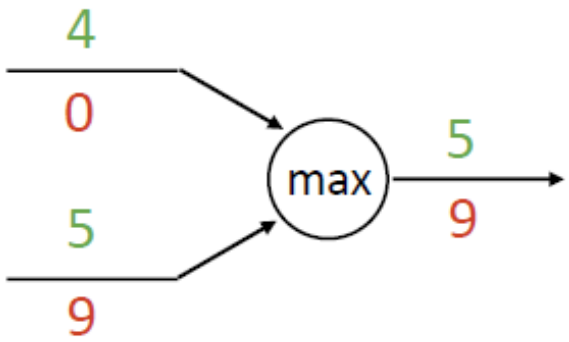
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router



Summary: Training Neural Networks

- In neural networks, complex expressions including forward pass and backward pass can be represented as **computational graphs**

— **Forward pass:** Computes outputs

— **Backward pass:** Computes gradients

- During the backward pass, each node receives **upstream gradients** and multiplies them by **local gradients** to compute **down stream gradients**