Multi-Layer Networks and Backpropagation Algorithm

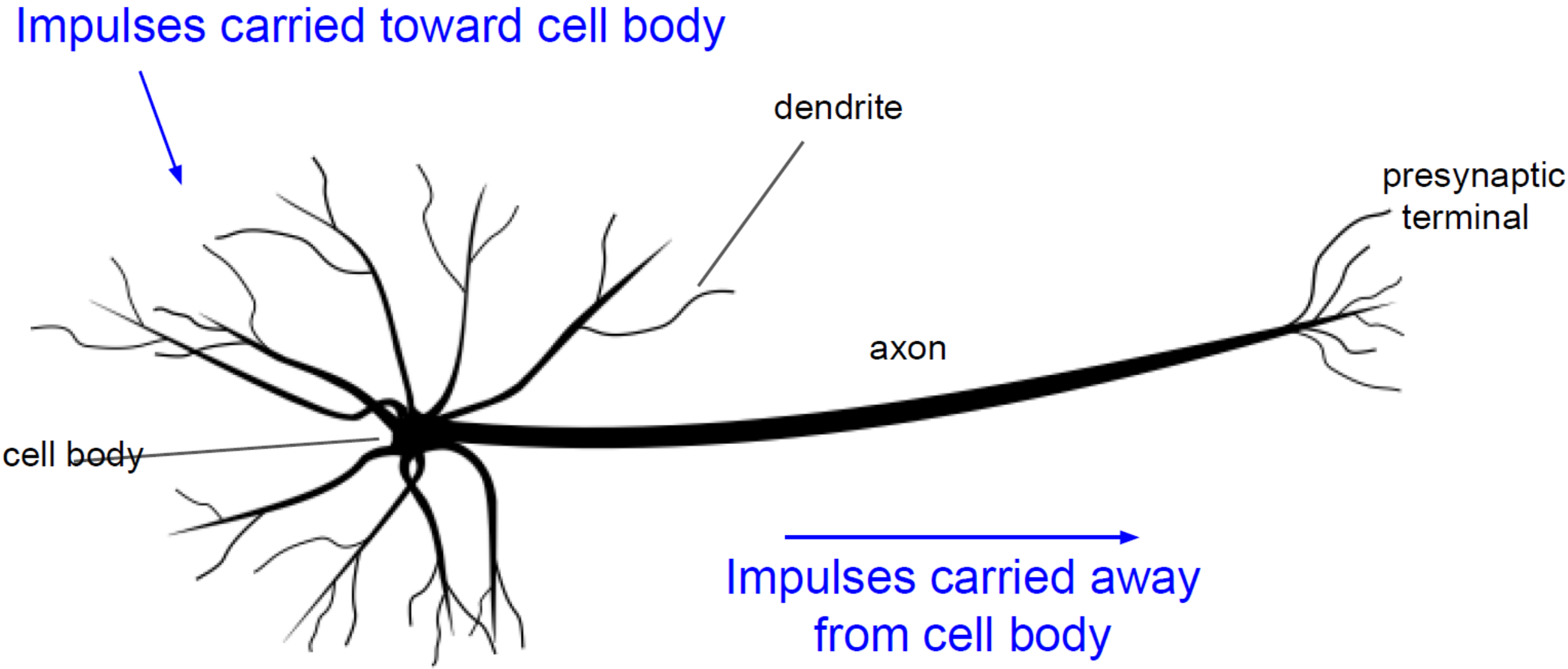
M. Soleymani
Sharif University of Technology
Fall 2017

Most slides have been adapted from Fei Fei Li lectures, cs231n, Stanford 2017 and some from Hinton lectures, “NN for Machine Learning” course, 2015.

Reasons to study neural computation

- Neuroscience: To understand how the brain actually works.
  - Its very big and very complicated and made of stuff that dies when you poke it around. So we need to use computer simulations.

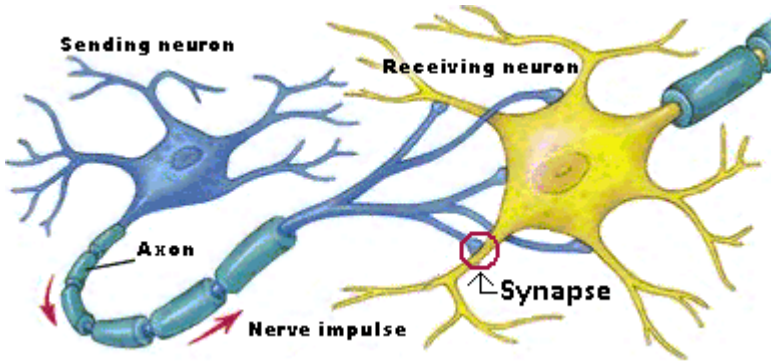
- AI: To solve practical problems by using novel learning algorithms inspired by the brain
  - Learning algorithms can be very useful even if they are not how the brain actually works.



A typical cortical neuron

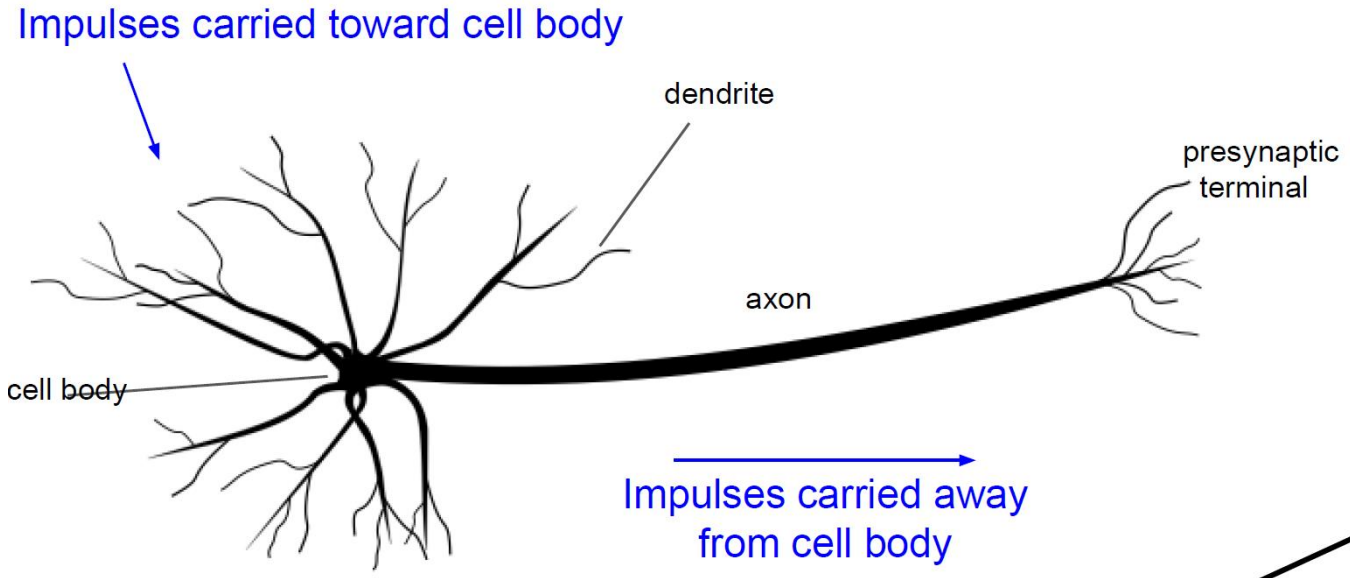
- Gross physical structure:
  - There is one axon that branches
  - There is a dendritic tree that collects input from other neurons.

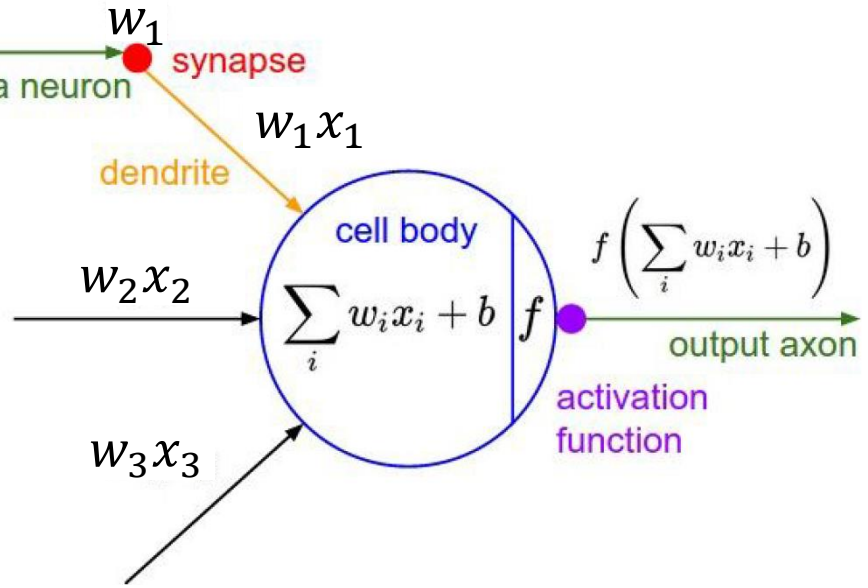
- Axons typically contact dendritic trees at synapses
  - A spike of activity in the axon causes charge to be injected into the post-synaptic neuron.

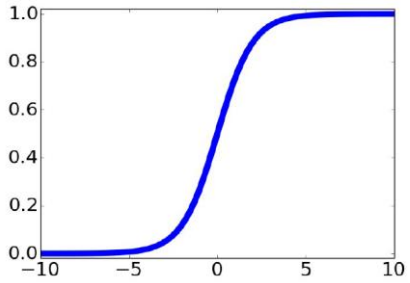


- Spike generation:
  - There is an axon hillock that generates outgoing spikes whenever enough charge has flowed in at synapses to depolarize the cell membrane.

A mathematical model for biological neurons







sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

How the brain works

- Each neuron receives inputs from other neurons
- The effect of each input line on the neuron is controlled by a synaptic weight
- The synaptic weights adapt so that the whole network learns to perform useful computations
  - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^{4} weights.
  - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.

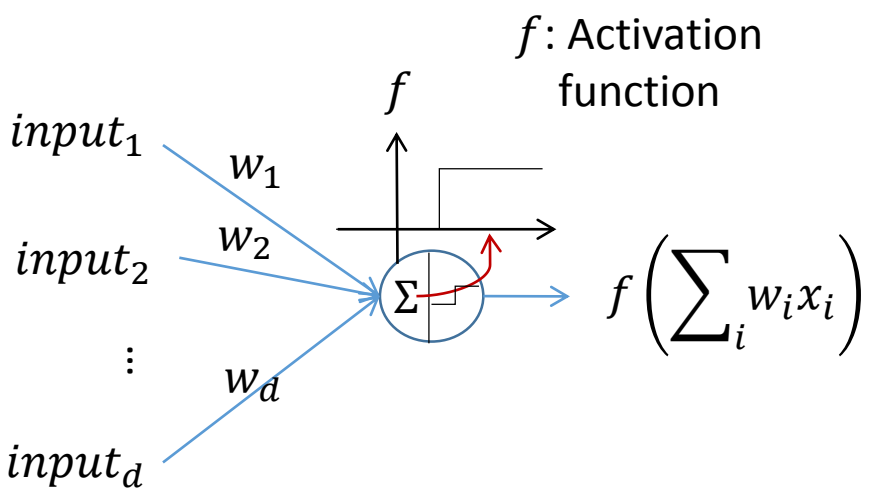
Be very careful with your brain analogies!

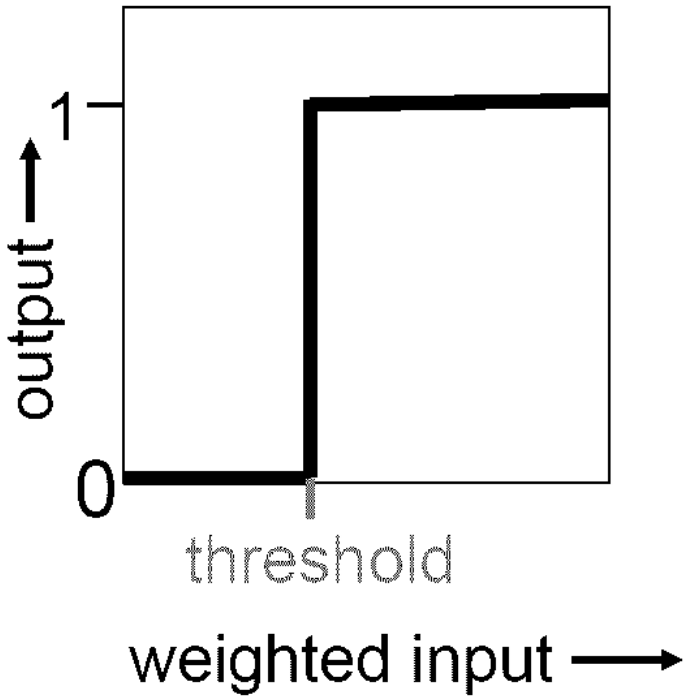
- **Biological Neurons:**
  - Many different types
  - Dendrites can perform complex non-linear computations
  - Synapses are not a single weight but a complex non-linear dynamical system
  - Rate code may not be adequate

[Dendritic Computation. London and Hausser]

Binary threshold neurons

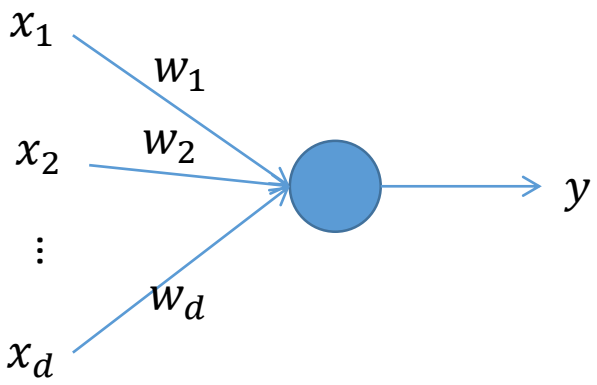
- McCulloch-Pitts (1943): influenced Von Neumann.
  - First compute a weighted sum of the inputs.
  - send out a spike of activity if the weighted sum exceeds a threshold.
  - McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition!



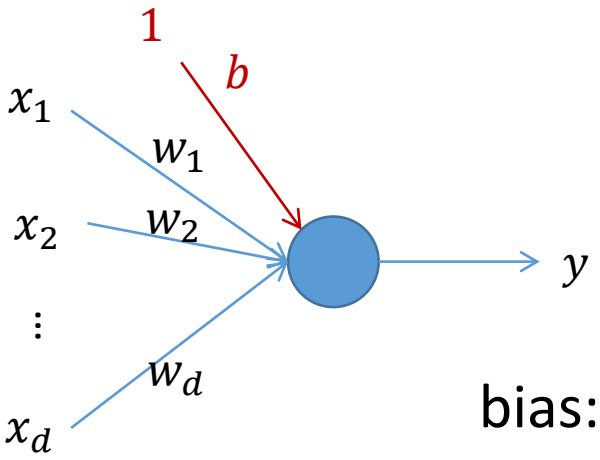


McCulloch-Pitts neuron: binary threshold

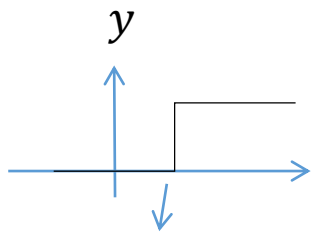
- Neuron, unit, or processing element:



Equivalent to

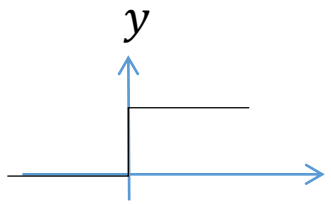


binary McCulloch-Pitts neuron



θ: activation threshold

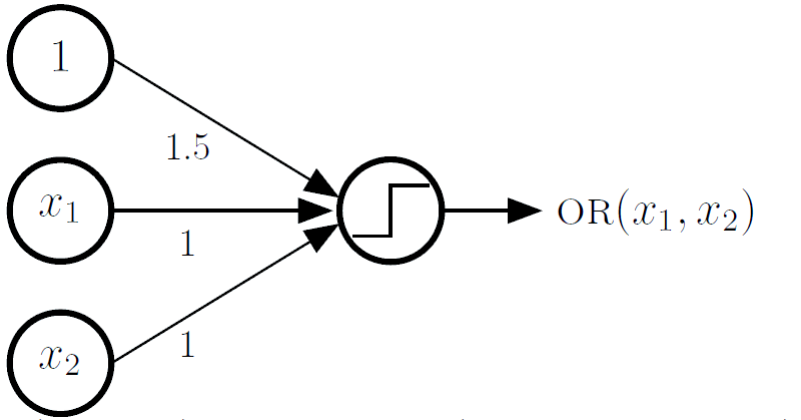
$$y = \begin{cases} 1, & z \geq \theta \\ 0, & z < \theta \end{cases}$$



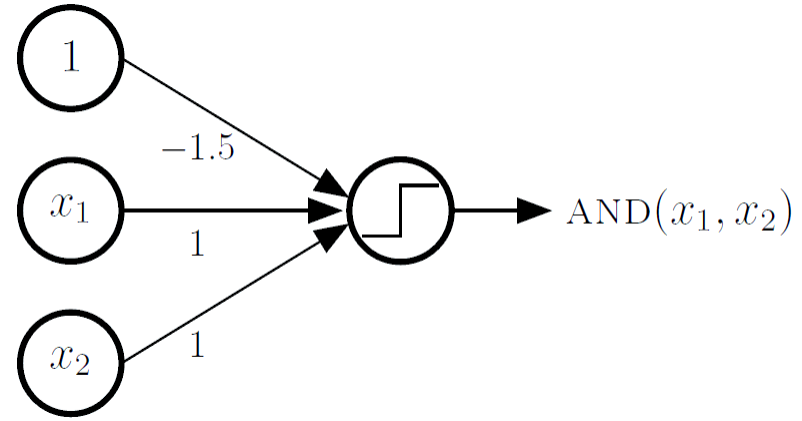
bias: $b = -\theta$

AND & OR networks

- For -1 and 1 inputs:



$$\mathrm{OR}(x_1, x_2) = \mathrm{sign}(x_1 + x_2 + 1.5)$$

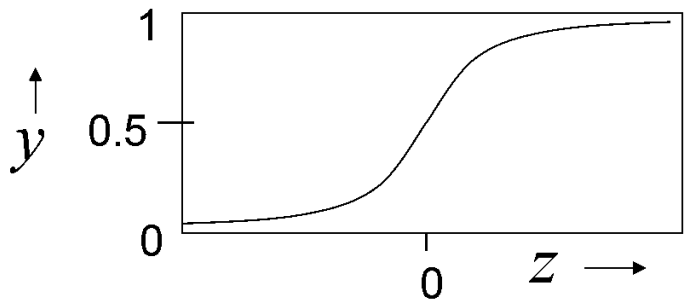


$$\mathrm{AND}(x_1, x_2) = \mathrm{sign}(x_1 + x_2 - 1.5)$$

Sigmoid neurons

- These give a real-valued output that is a smooth and bounded function of their total input.

- Typically they use the logistic function
  - They have nice derivatives.

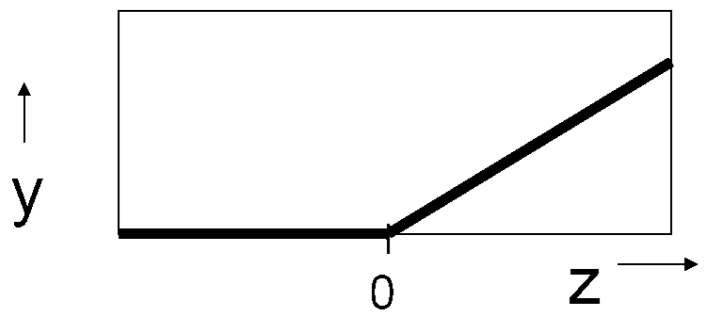


Rectified Linear Units (ReLU)

- They compute a linear weighted sum of their inputs.
- The output is a non-linear function of the total input.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$

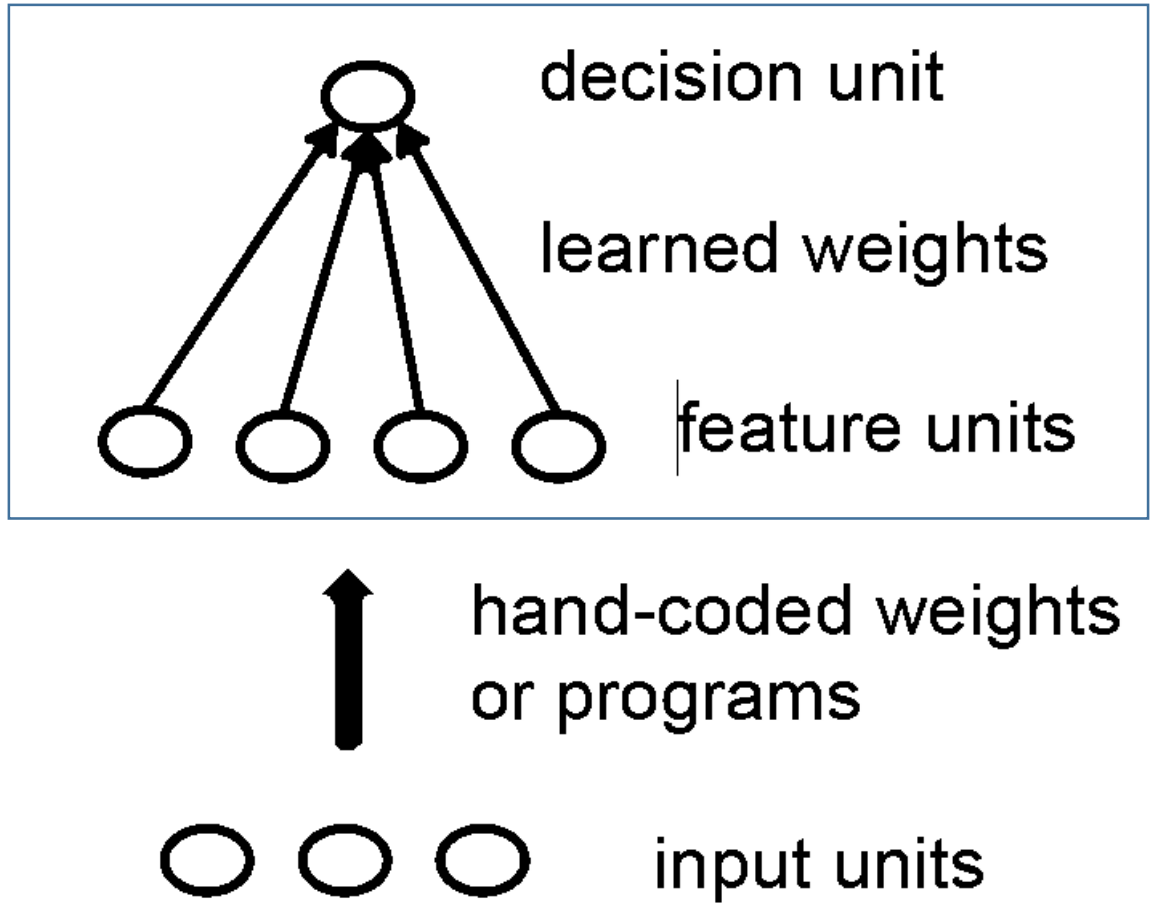


Adjusting weights

- Types of single layer networks:
  - Perceptron (Rosenblatt, 1962)
  - ADALINE (Widrow and Hoff, 1960)

The standard Perceptron architecture

- Learn how to weight each of the feature activations to get desirable outputs.
- If output is above some threshold, decide that the input vector is a positive example of the target class.



The perceptron convergence procedure

- Perceptron trains binary output neurons as classifiers
- Pick training cases (until convergence):
  - If the output unit is correct, leave its weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to it.
  - If the output unit incorrectly outputs a 1, subtract the input vector from it.

- This is guaranteed to find a set of weights that gets the right answer for all the training cases if any such set exists.

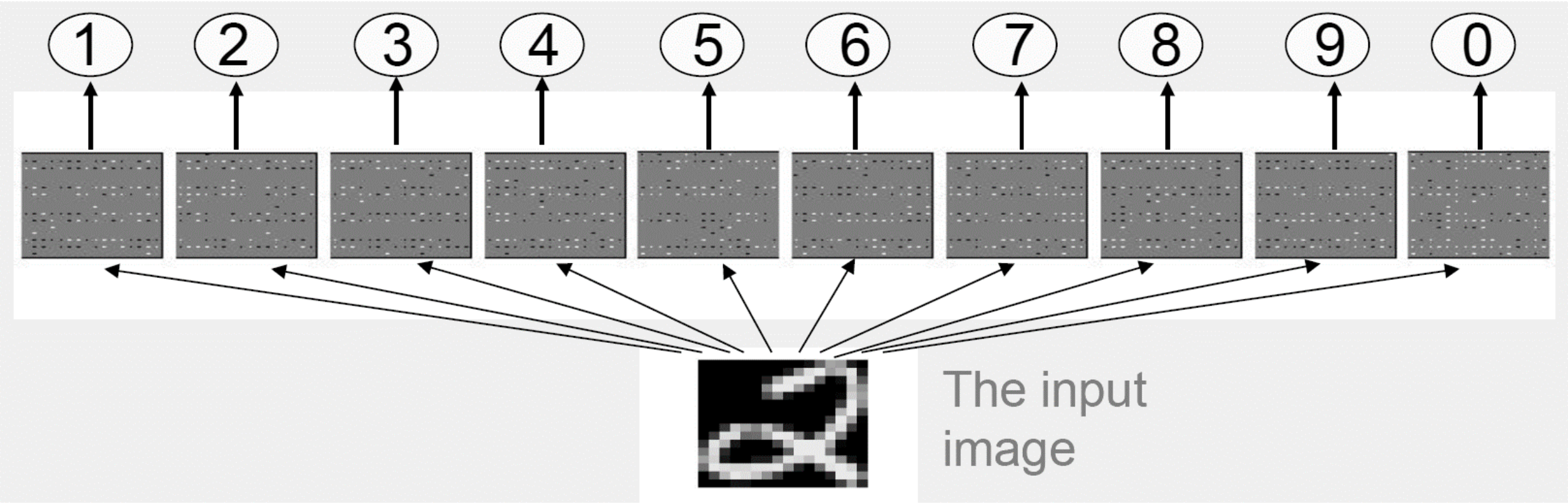
Adjusting weights

- Weight update for a training pair $(\boldsymbol{x}^{(n)}, y^{(n)})$:
  - **Perceptron**: If $sign(\boldsymbol{w}^T \boldsymbol{x}^{(n)}) \neq y^{(n)}$ then $\Delta \boldsymbol{w} = \boldsymbol{x}^{(n)} y^{(n)}$ else $\Delta \boldsymbol{w} = \boldsymbol{0}$
  - **ADALINE**: $\Delta \boldsymbol{w} = \eta (y^{(n)} - \boldsymbol{w}^T \boldsymbol{x}^{(n)}) \boldsymbol{x}^{(n)}$
    - Widrow-Hoff, LMS, or delta rule

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \nabla E_n(\boldsymbol{w}^t)$$

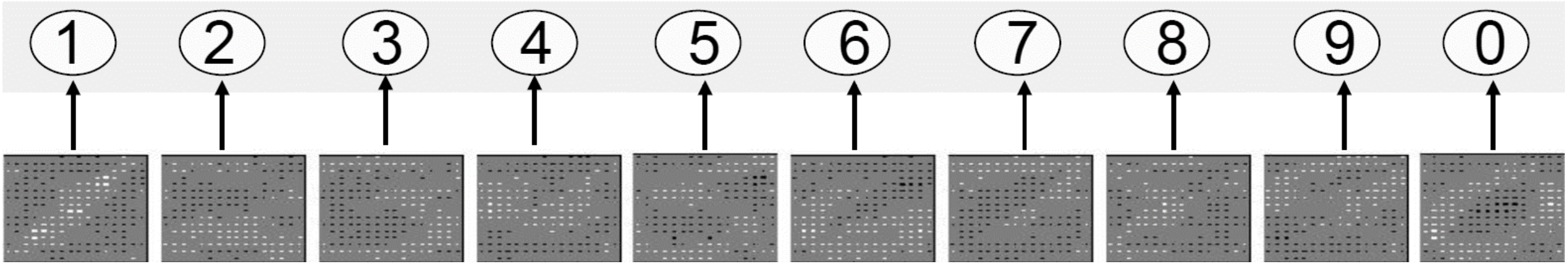
$$E_n(\boldsymbol{w}) = \left(y^{(n)} - \boldsymbol{w}^T \boldsymbol{x}^{(n)}\right)^2$$

How to learn the weights: multi class example



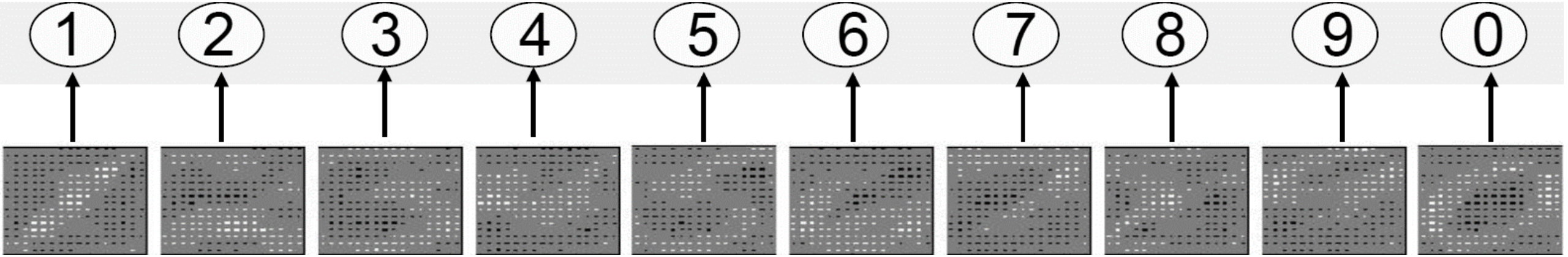
How to learn the weights: multi class example

- If correct: no change
- If wrong:
  - lower score of the wrong answer (by removing the input from the weight vector of the wrong answer)
  - raise score of the target (by adding the input to the weight vector of the target class)



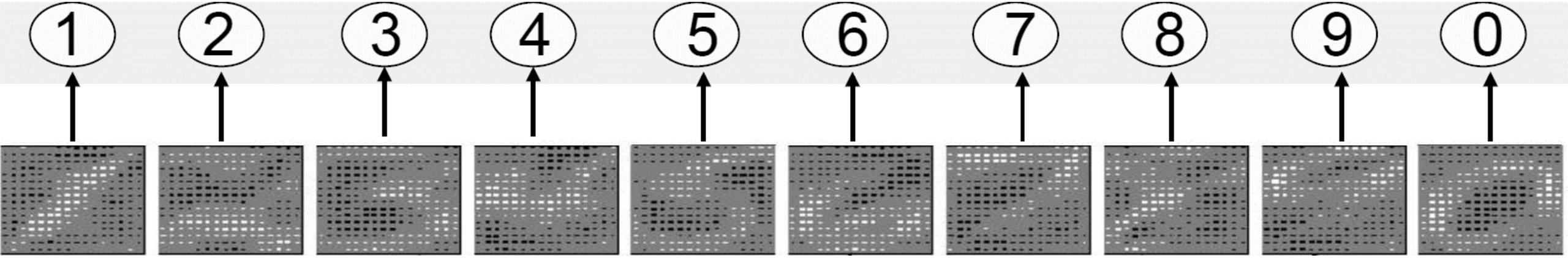
How to learn the weights: multi class example

- If correct: no change
- If wrong:
  - lower score of the wrong answer (by removing the input from the weight vector of the wrong answer)
  - raise score of the target (by adding the input to the weight vector of the target class)



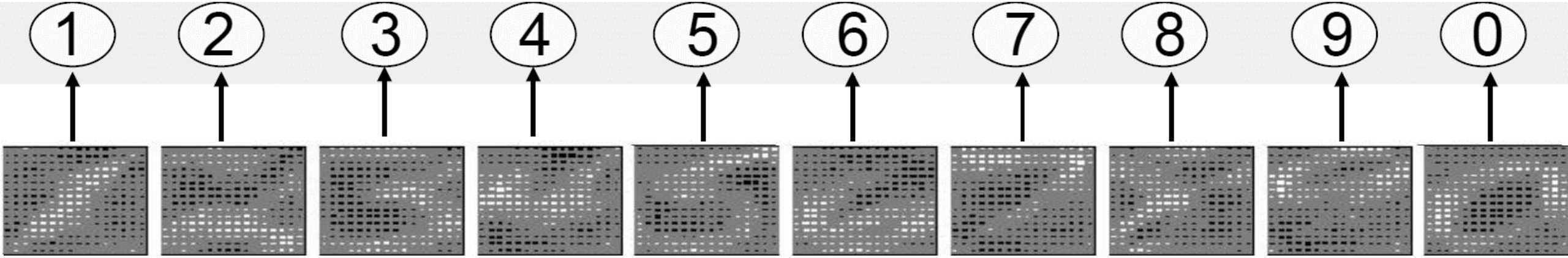
How to learn the weights: multi class example

- If correct: no change
- If wrong:
  - lower score of the wrong answer (by removing the input from the weight vector of the wrong answer)
  - raise score of the target (by adding the input to the weight vector of the target class)



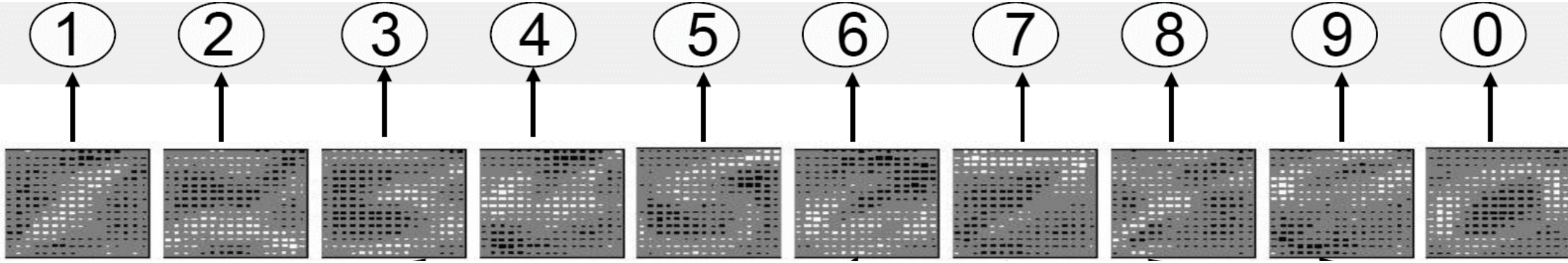
How to learn the weights: multi class example

- If correct: no change
- If wrong:
  - lower score of the wrong answer (by removing the input from the weight vector of the wrong answer)
  - raise score of the target (by adding the input to the weight vector of the target class)



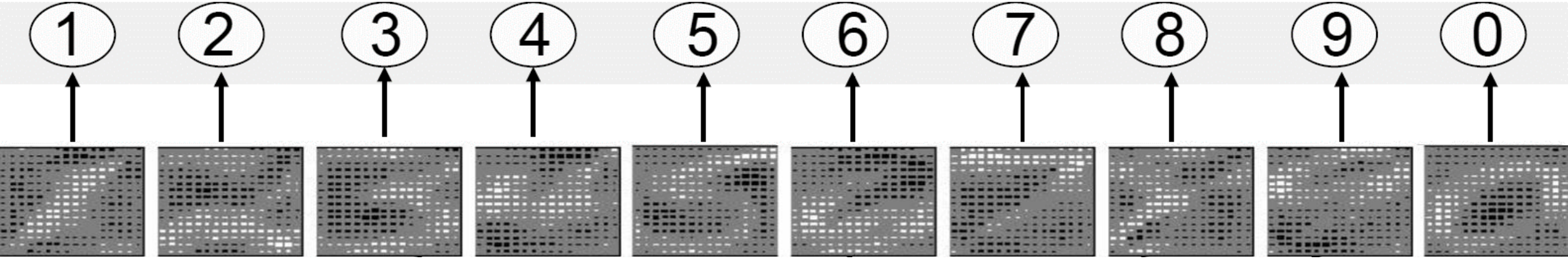
How to learn the weights: multi class example

- If correct: no change
- If wrong:
  - lower score of the wrong answer (by removing the input from the weight vector of the wrong answer)
  - raise score of the target (by adding the input to the weight vector of the target class)



How to learn the weights: multi class example

- If correct: no change
- If wrong:
  - lower score of the wrong answer (by removing the input from the weight vector of the wrong answer)
  - raise score of the target (by adding the input to the weight vector of the target class)



Single layer networks as template matching

- Weights for each class as a template (or sometimes also called a prototype) for that class.
  - The winner is the most similar template.

- The ways in which hand-written digits vary are much too complicated to be captured by simple template matches of whole shapes.

- To capture all the allowable variations of a digit we need to learn the features that it is composed of.

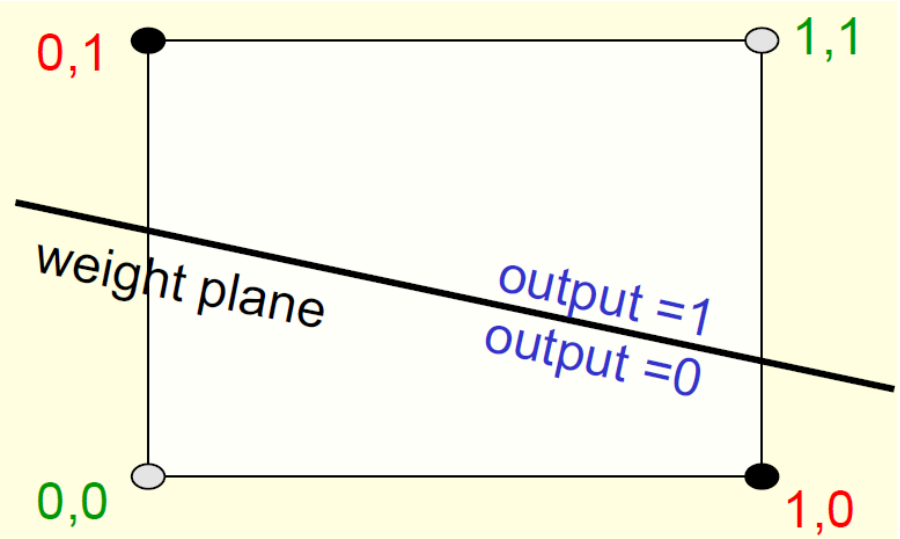
The history of perceptrons

- They were popularised by Frank Rosenblatt in the early 1960's.
  - They appeared to have a very powerful learning algorithm.
  - Lots of grand claims were made for what they could learn to do.

- In 1969, Minsky and Papert published a book called "Perceptrons" that analyzed what they could do and showed their limitations.
  - Many people thought these limitations applied to all neural network models.

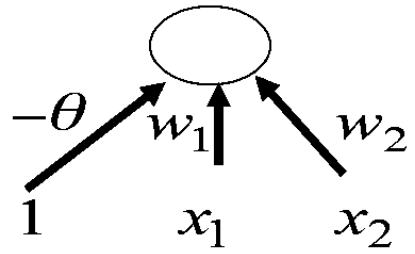
What binary threshold neurons cannot do

- A binary threshold output unit cannot even tell if two single bit features are the same!
- A geometric view of what binary threshold neurons cannot do
- The positive and negative cases cannot be separated by a plane



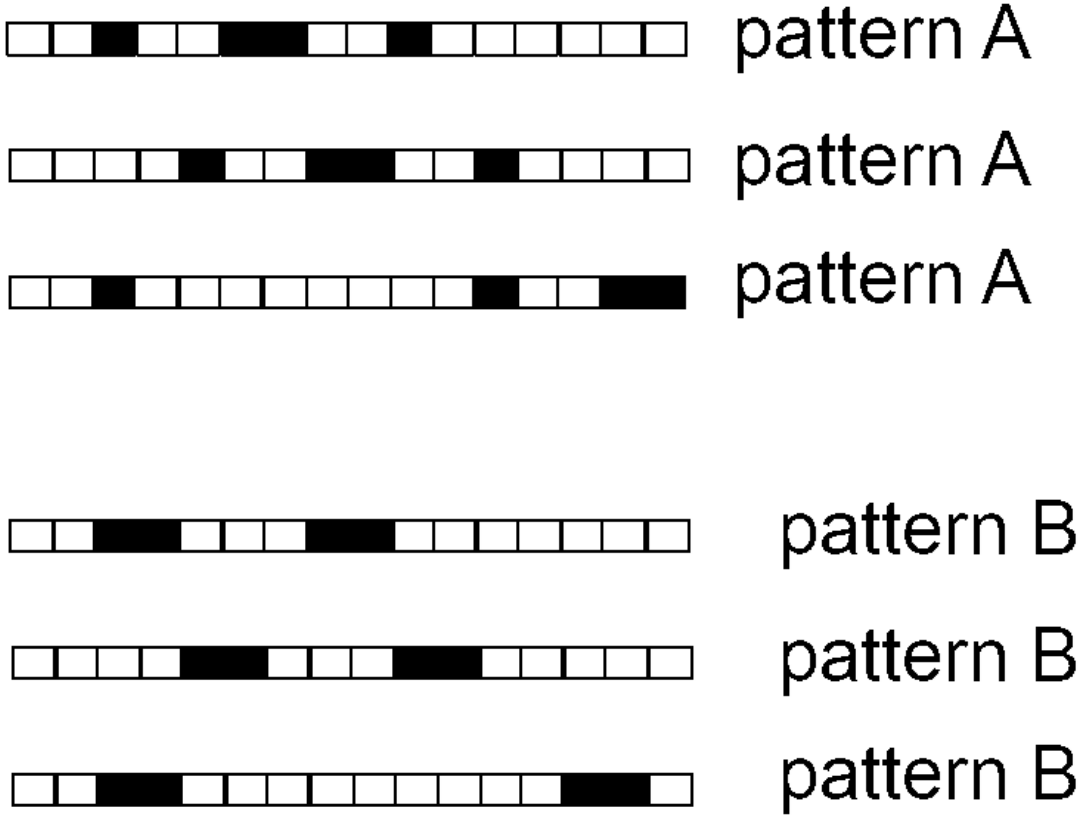
What binary threshold neurons cannot do

- Positive cases (same): (1,1)->1; (0,0)->1
- Negative cases (different): (1,0)->0; (0,1)->0
- The four input-output pairs give four inequalities that are impossible to satisfy:
  - $w1 + w2 \geq \theta$
  - $0 \geq \theta$
  - $w1 < \theta$
  - $w2 < \theta$



Discriminating simple patterns under translation with wrap-around

- Suppose we just use pixels as the features.
- binary decision unit cannot discriminate patterns with the same number of on pixels
  - if the patterns can translate with wrap-around!



Sketch of a proof

- For pattern A, use training cases in all possible translations.
  - Each pixel will be activated by 4 different translations of pattern A.
  - So the total input received by the decision unit over all these patterns will be four times the sum of all the weights.
- For pattern B, use training cases in all possible translations.
  - Each pixel will be activated by 4 different translations of pattern B.
  - So the total input received by the decision unit over all these patterns will be four times the sum of all the weights.
- But to discriminate correctly, every single case of pattern A must provide more input to the decision unit than every single case of pattern B.
- This is impossible if the sums over cases are the same.

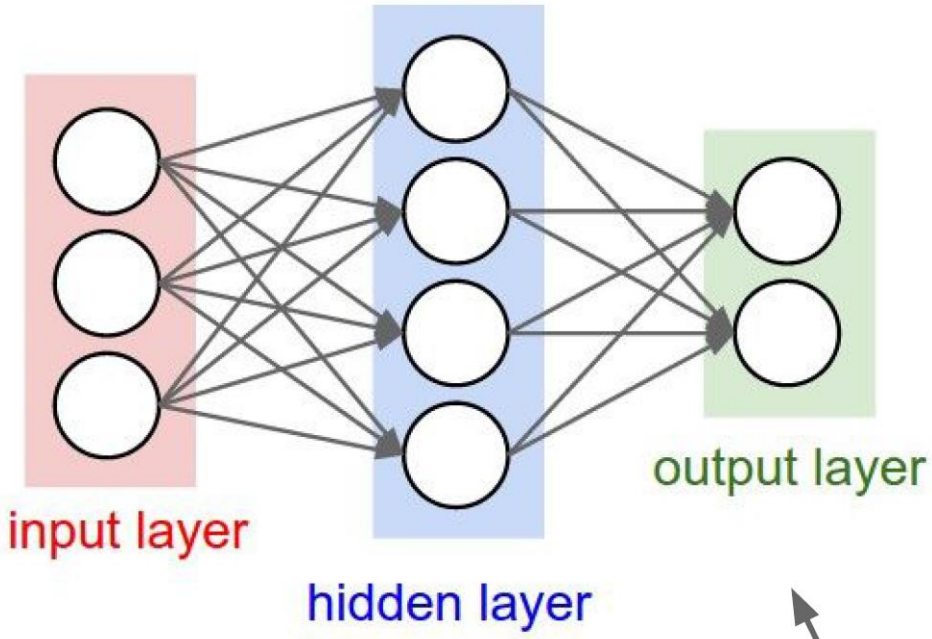
Networks with hidden units

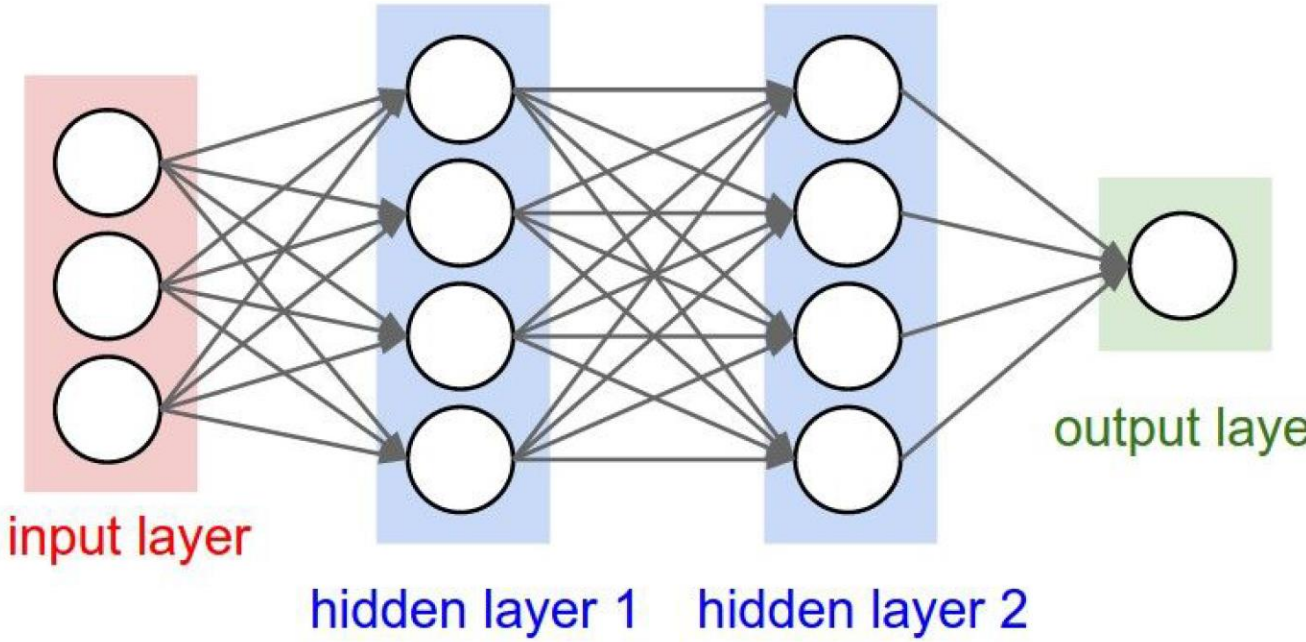
- Networks without hidden units are very limited in the input-output mappings they can learn to model.
  - More layers of linear units do not help. Its still linear.
  - Fixed output non-linearities are not enough.

- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?

Feed-forward neural networks

- Also called **Multi-Layer Perceptron (MLP)**





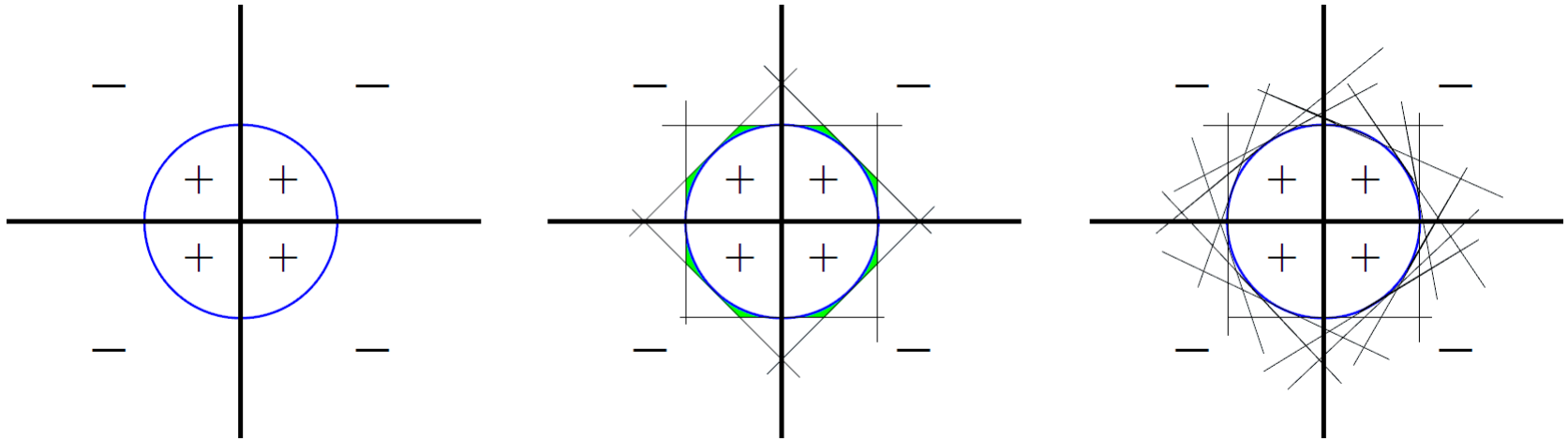
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

General approximator

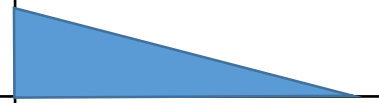
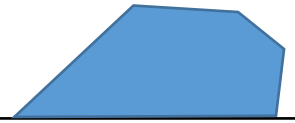
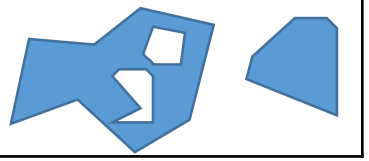
- If the decision boundary is smooth, then a 3-layer network (i.e. 2 hidden layer) can come arbitrarily close to the target classifier



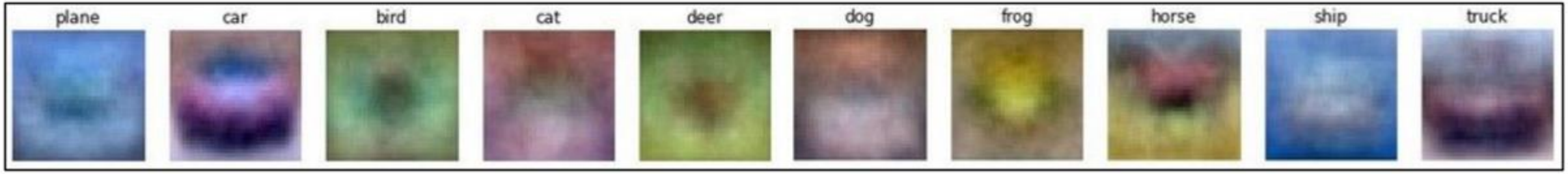
MLP with Different Number of Layers

MLP with unit step activation function

Decision region found by an output unit.

Structure	Type of Decision Regions	Interpretation	Example of region
Single Layer (no hidden layer)	Half space	Region found by a hyper-plane	
Two Layer (one hidden layer)	Polyhedral (open or closed) region	Intersection of half spaces	
Three Layer (two hidden layers)	Arbitrary regions	Union of polyhedrals	

Beyond linear models



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

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

Beyond linear models

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

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

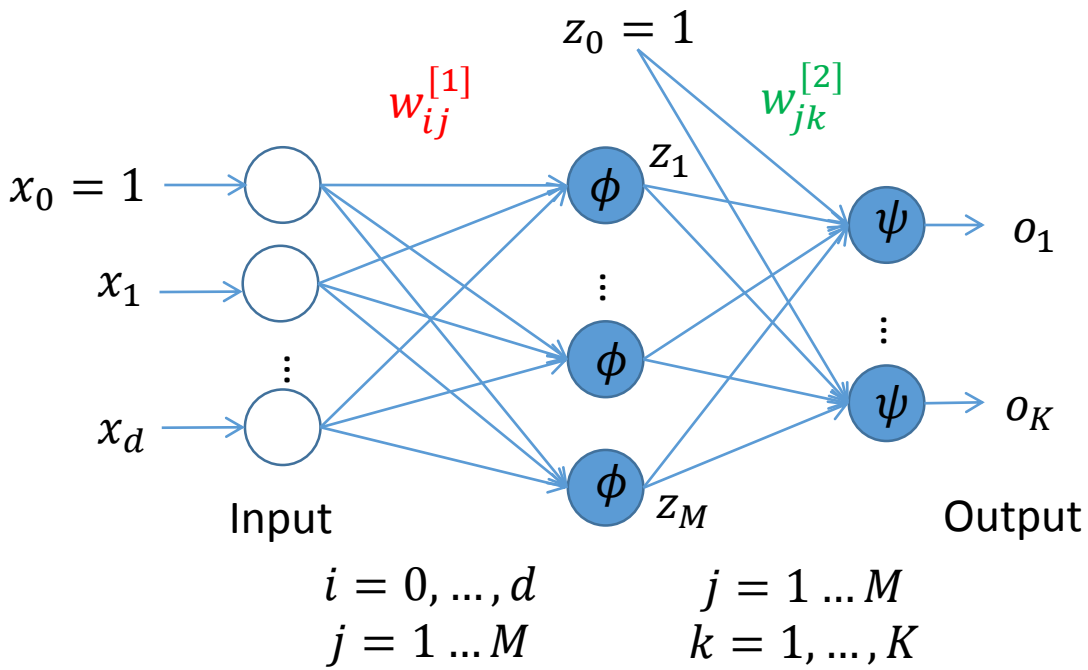
or 3-layer Neural Network

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

MLP with single hidden layer

- Two-layer MLP (Number of layers of adaptive weights is counted)

$$o_k(\boldsymbol{x}) = \psi\left(\sum_{j=0}^{M} w_{jk}^{[2]} z_j\right) \Rightarrow o_k(\boldsymbol{x}) = \psi\left(\sum_{j=0}^{M} w_{jk}^{[2]} \phi\left(\sum_{i=0}^{d} w_{ij}^{[1]} x_i\right)\right)$$



learns to extract features

- MLP with one hidden layer is a generalized linear model:
  - $o_k(\boldsymbol{x}) = \psi\left(\sum_{j=1}^{M} w_{jk}^{[2]} f_j(\boldsymbol{x})\right)$
  - $f_j(\boldsymbol{x}) = \phi\left(\sum_{i=0}^{d} w_{ji}^{[1]} x_i\right)$
  - The form of the nonlinearity (basis functions f_j) is adapted from the training data (not fixed in advance)
    - f_j is defined based on parameters which can be also adapted during training
- Thus, we don't need expert knowledge or time consuming tuning of hand-crafted features

Deep networks

- Deeper networks (with multiple hidden layers) can work better than a single-hidden-layer networks is an empirical observation
  - despite the fact that their representational power is equal.

- In practice usually 3-layer neural networks will outperform 2-layer nets, but going even deeper may not help much more.
  - This is in stark contrast to Convolutional Networks

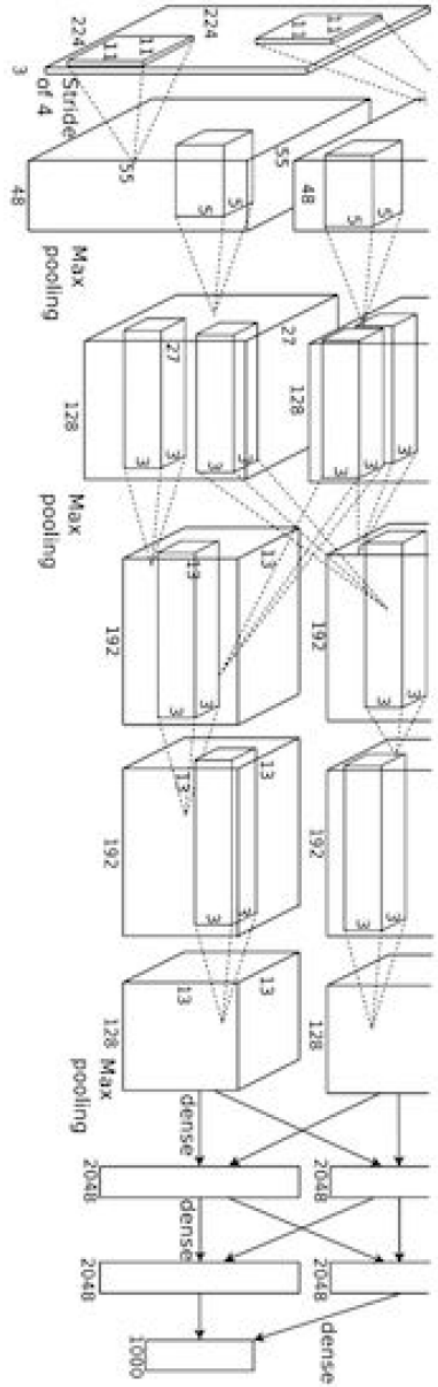
How to adjust weights for multi layer networks?

- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?
  - We need an efficient way of adapting all the weights, not just the last layer.
  - Learning the weights going into hidden units is equivalent to learning features.
  - This is difficult because nobody is telling us directly what the hidden units should do.

input image

weights

loss



Gradient descent

- We want $\nabla_{\boldsymbol{W}} L(\boldsymbol{W})$
- Numerical gradient:
  - slow :(
  - approximate :(
  - easy to write :)
- Analytic gradient:
  - fast :)
  - exact :)
  - error-prone :(
- In practice: Derive analytic gradient, check your implementation with numerical gradient

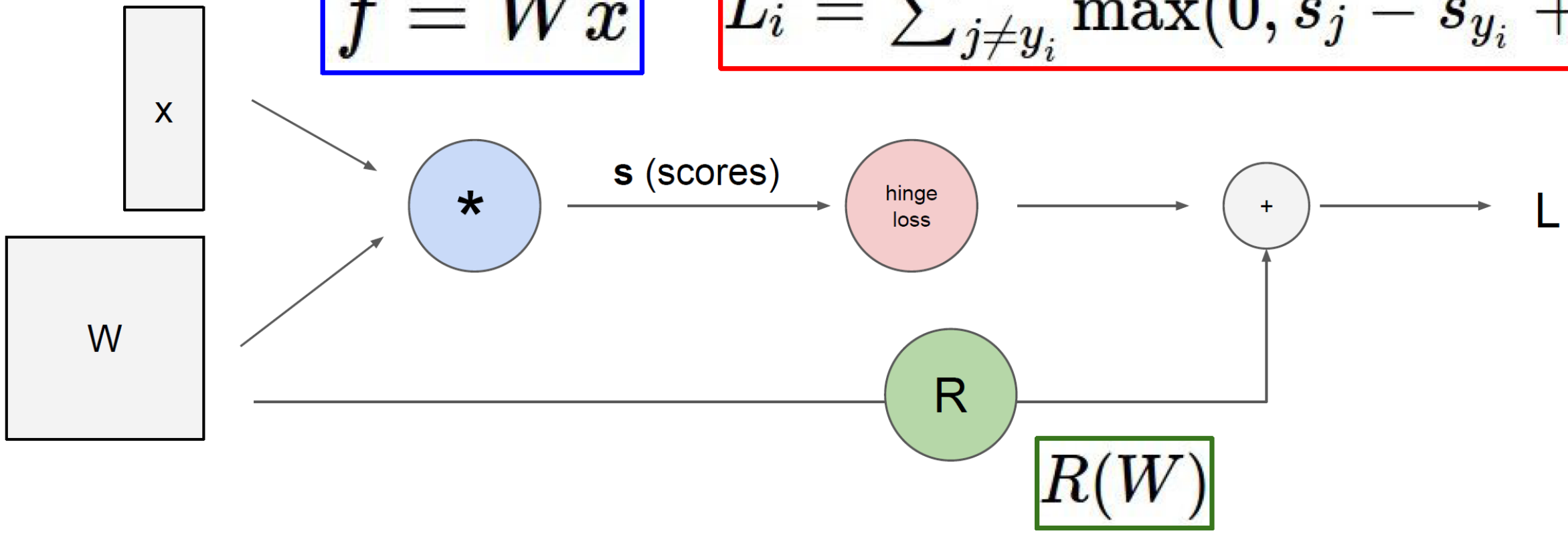
Training multi-layer networks

- Backpropagation
  - Training algorithm that is used to adjust weights in multi-layer networks (based on the training data)
  - The backpropagation algorithm is based on gradient descent
  - Use chain rule and dynamic programming to efficiently compute gradients

Computational graphs

$$f = Wx$$

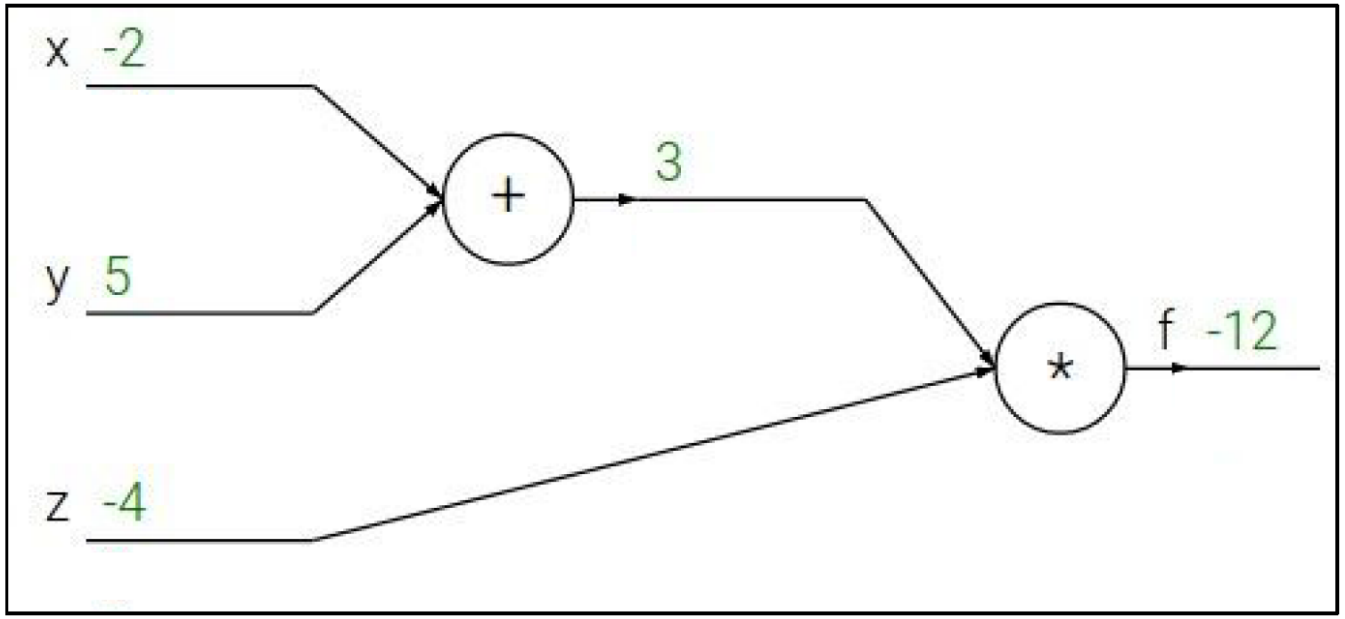
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



Backpropagation: a simple example

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

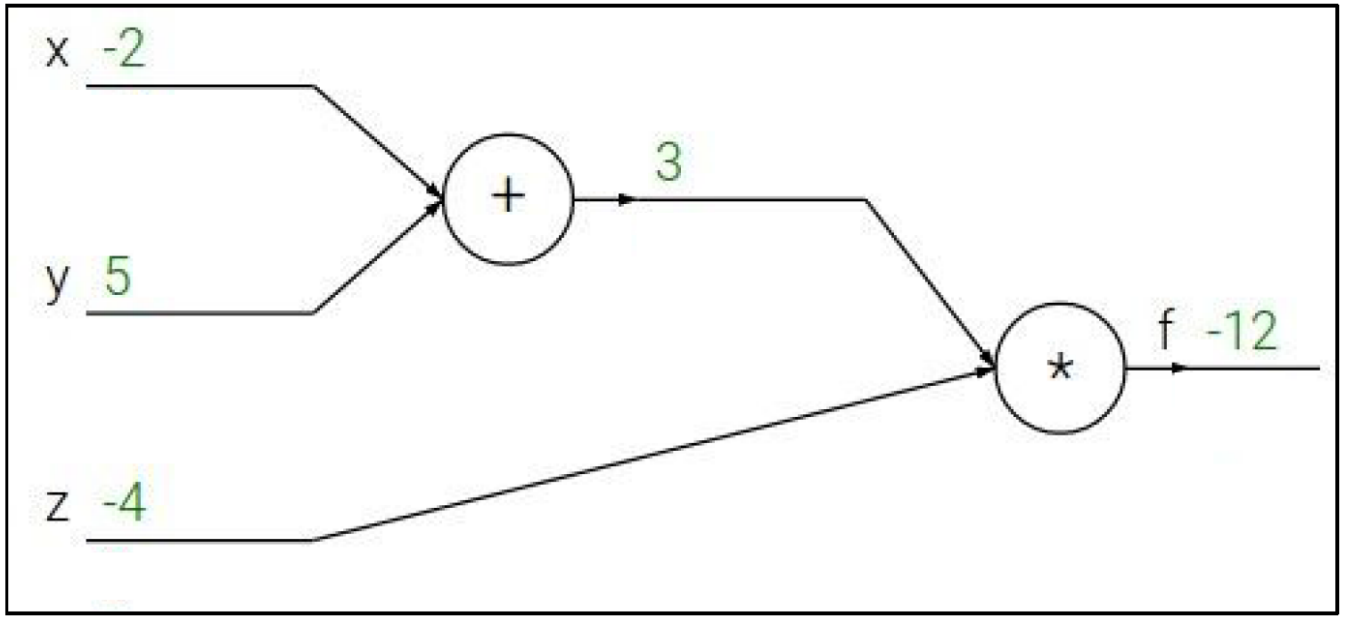
e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

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

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



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

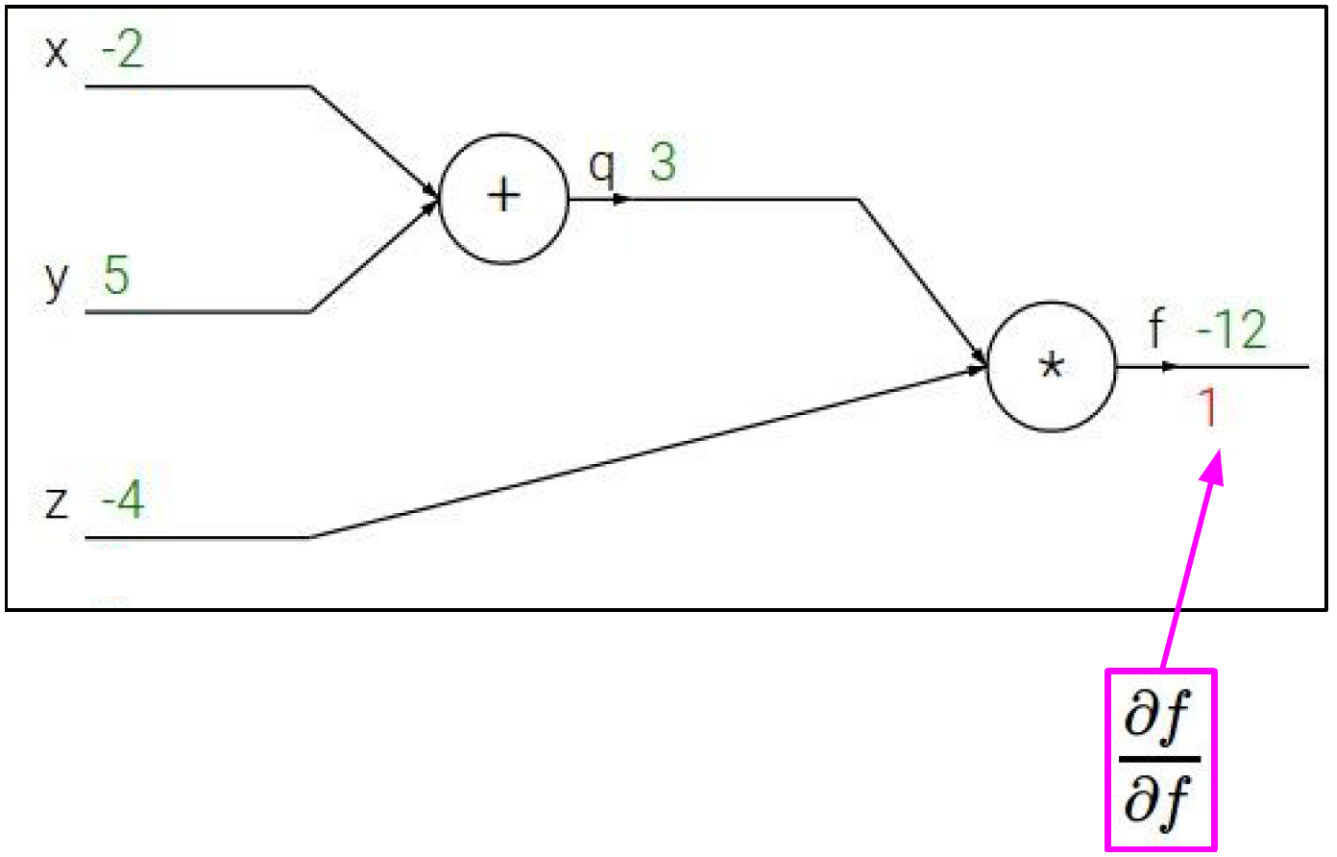
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Backpropagation: a simple example

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

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



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

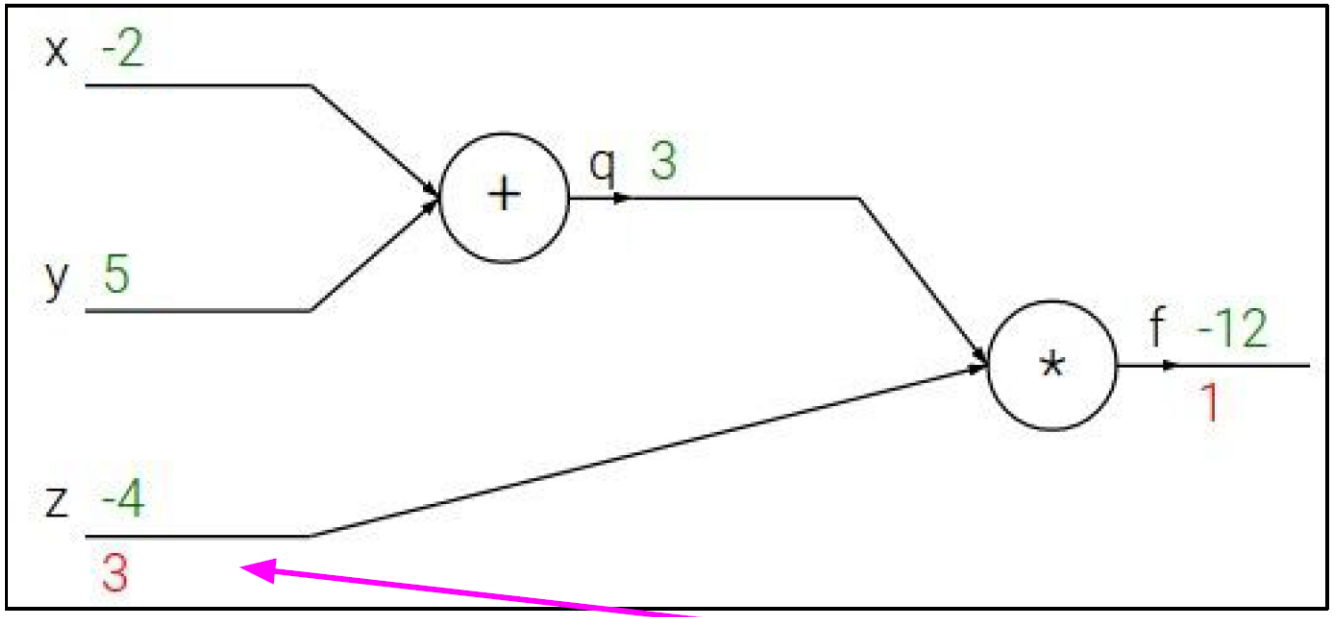
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Backpropagation: a simple example

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

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



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

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

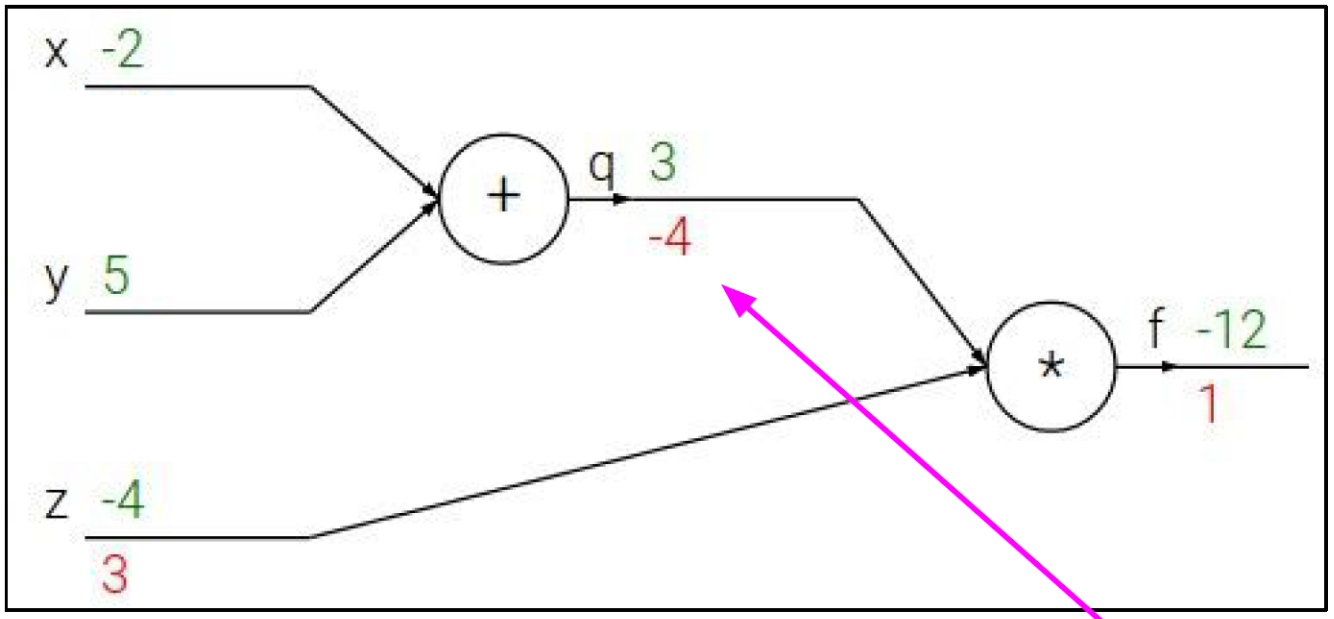
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: a simple example

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

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



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

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

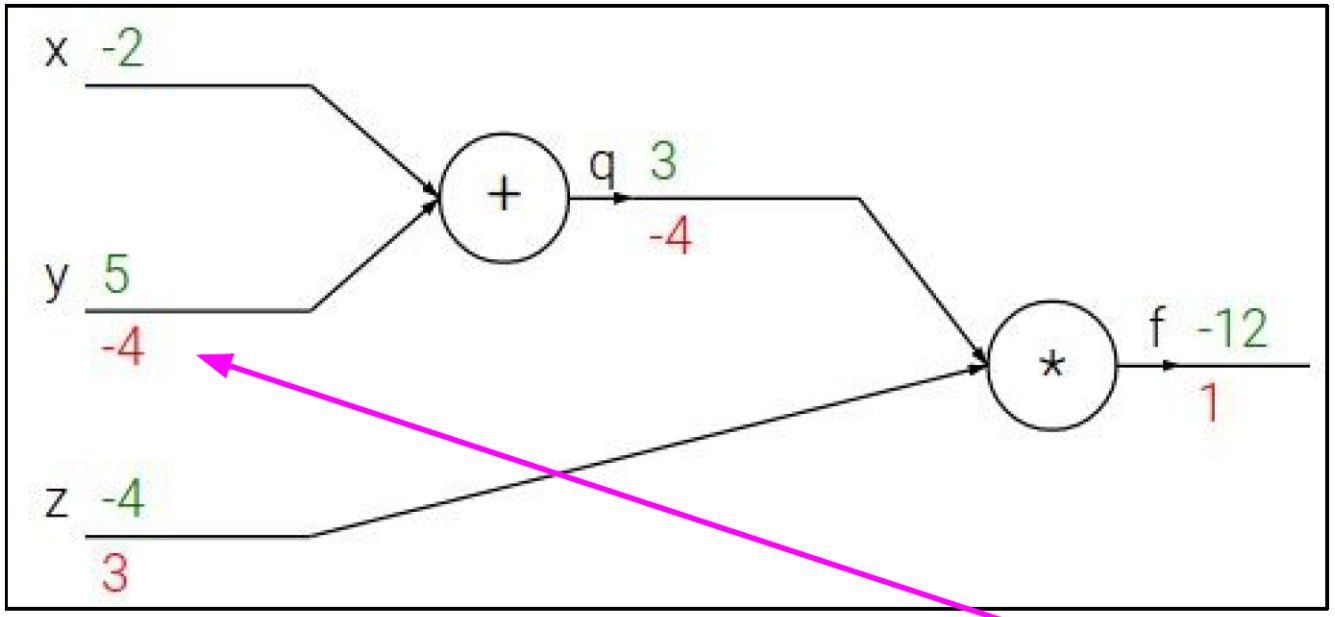
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: a simple example

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

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

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

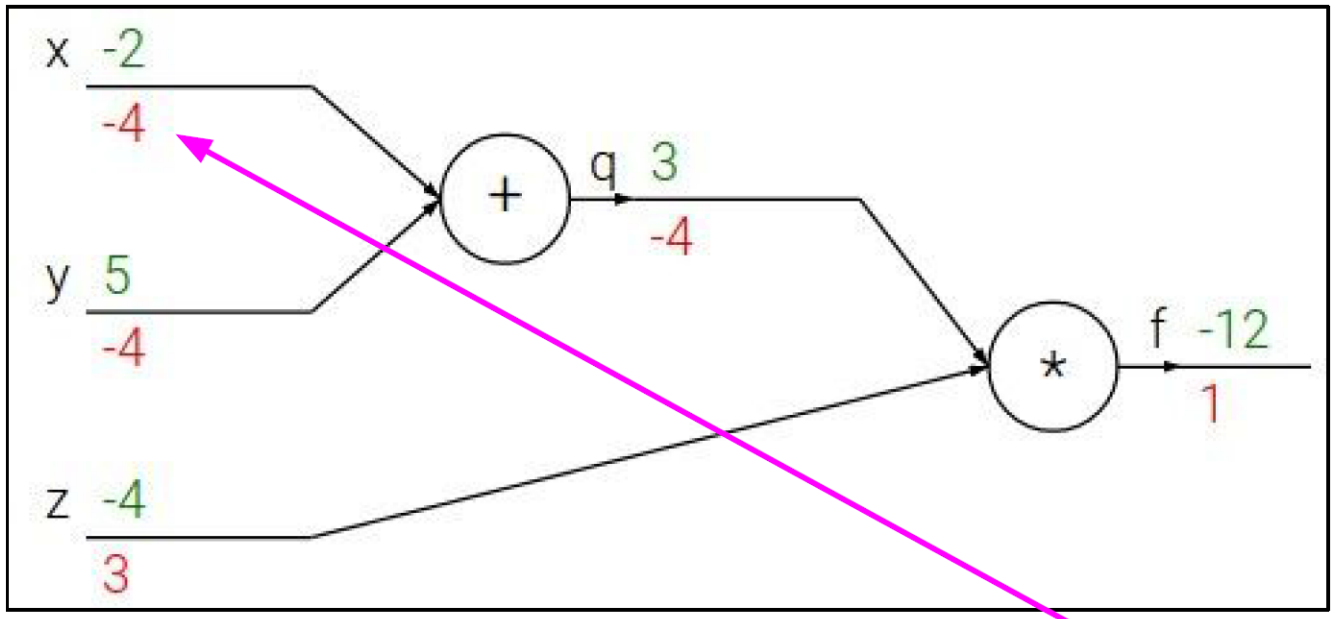
Chain rule:

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

Backpropagation: a simple example

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

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



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

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

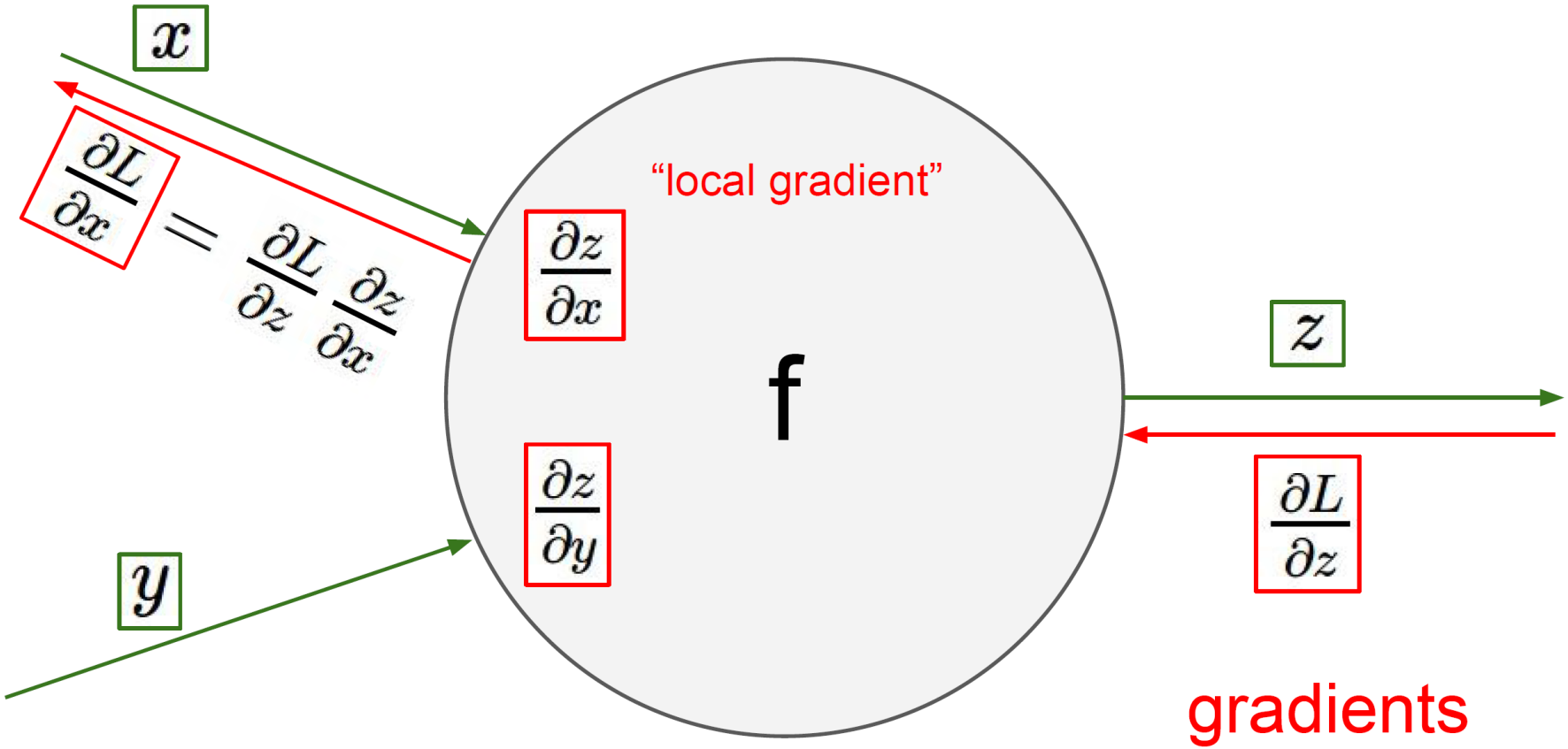
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

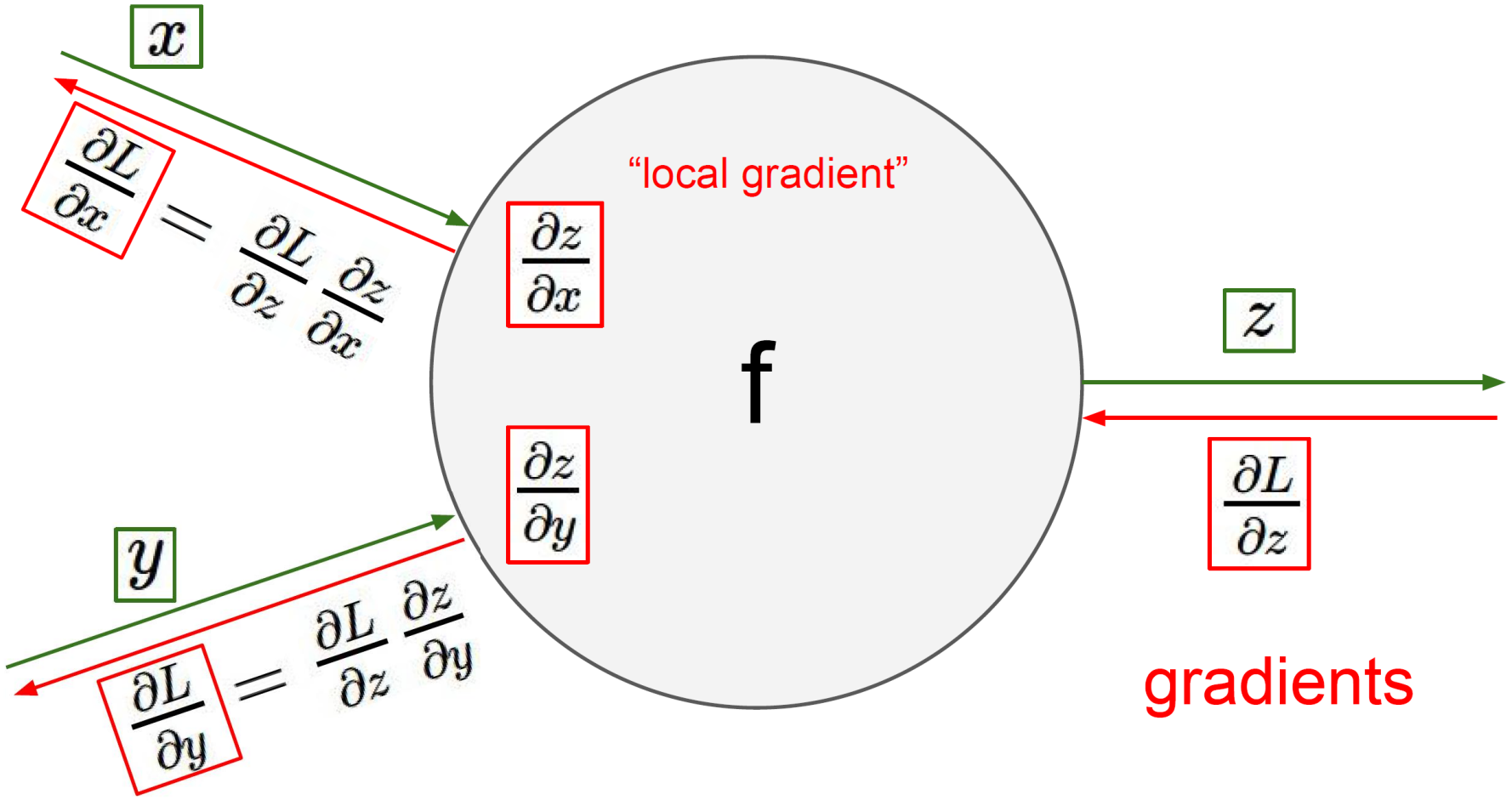
Chain rule:

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

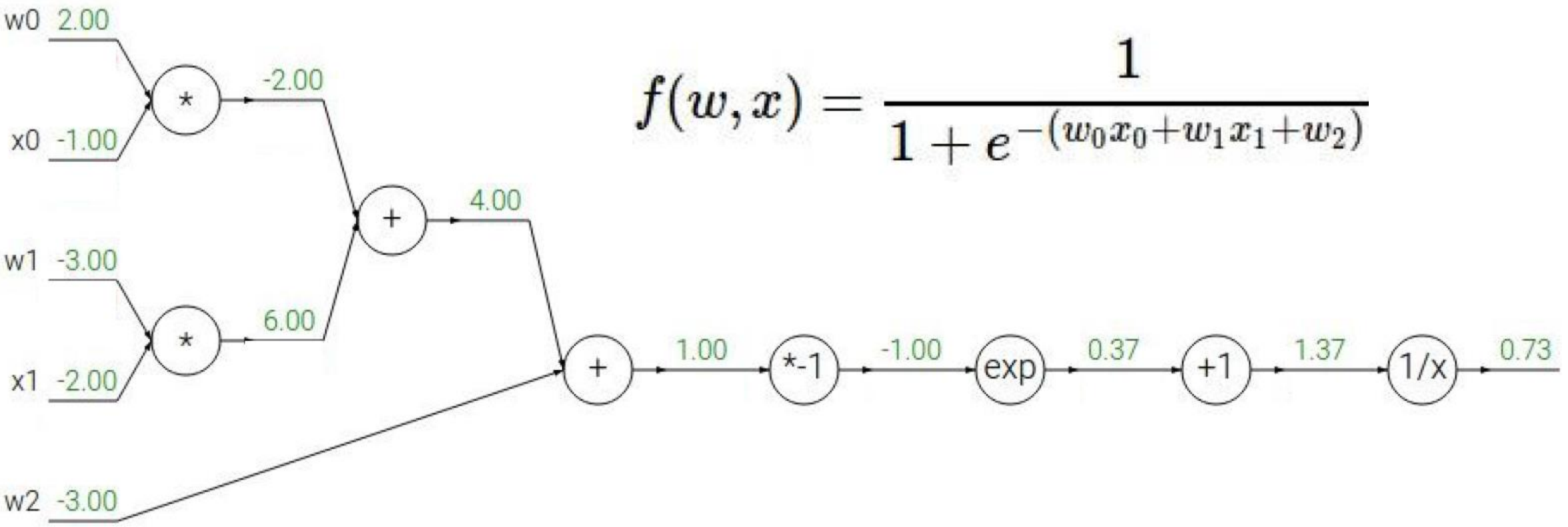
How to propagate the gradients backward



How to propagate the gradients backward

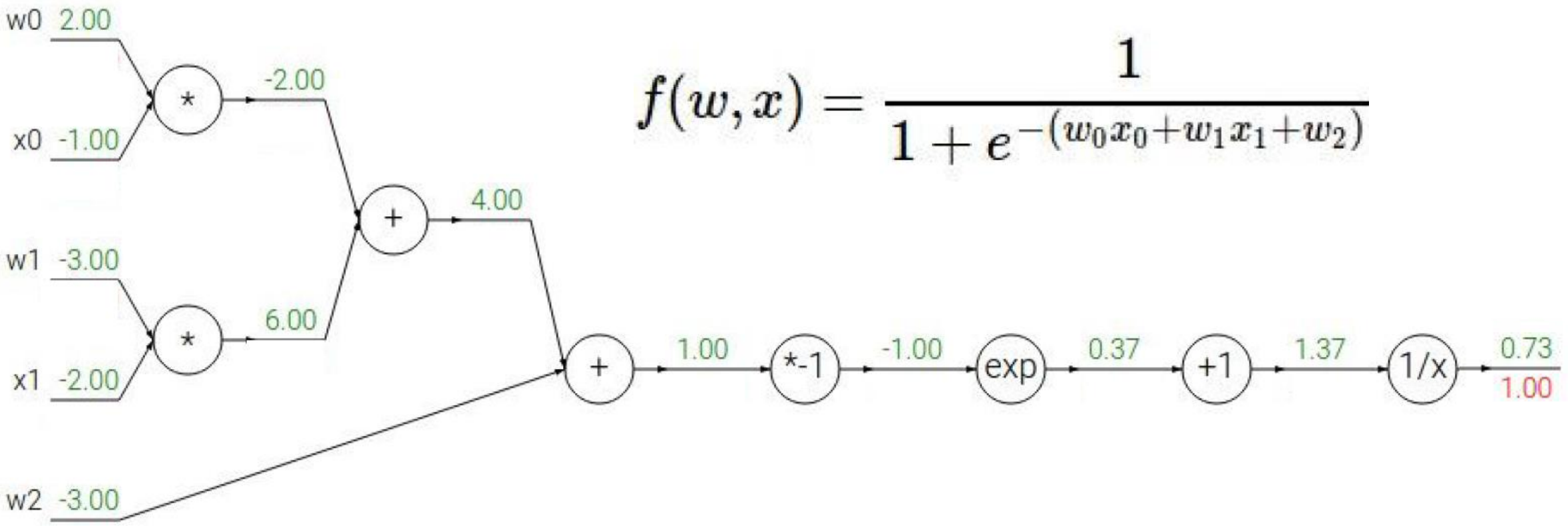


Another example



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

Another example



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

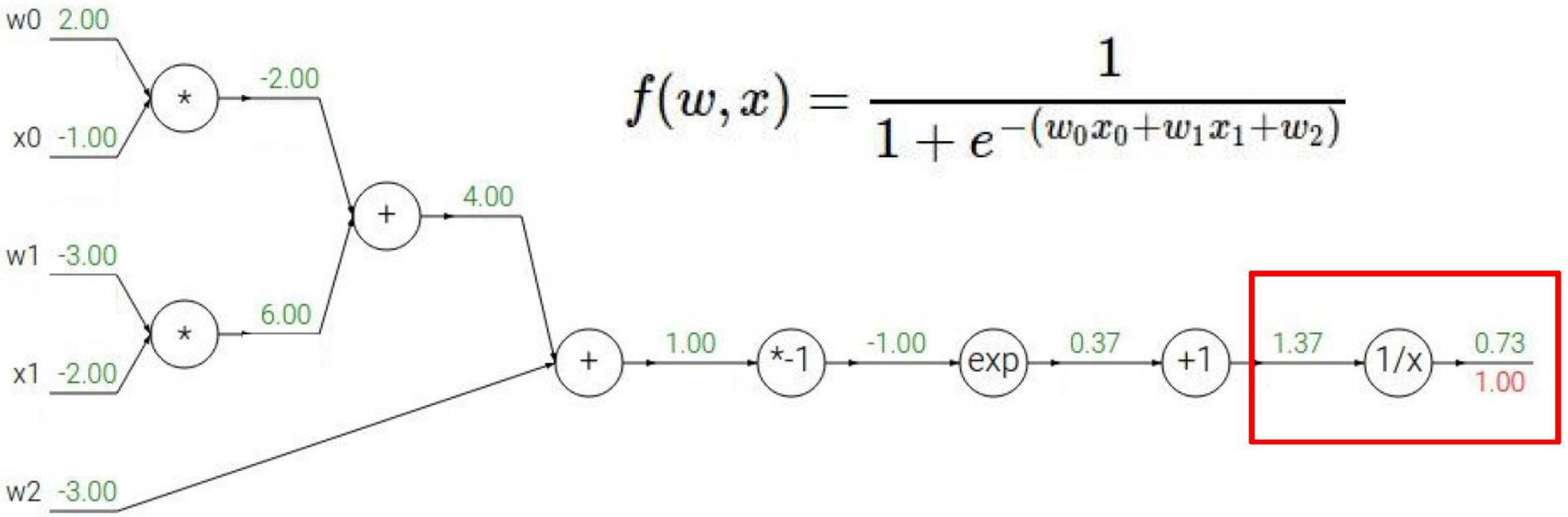
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

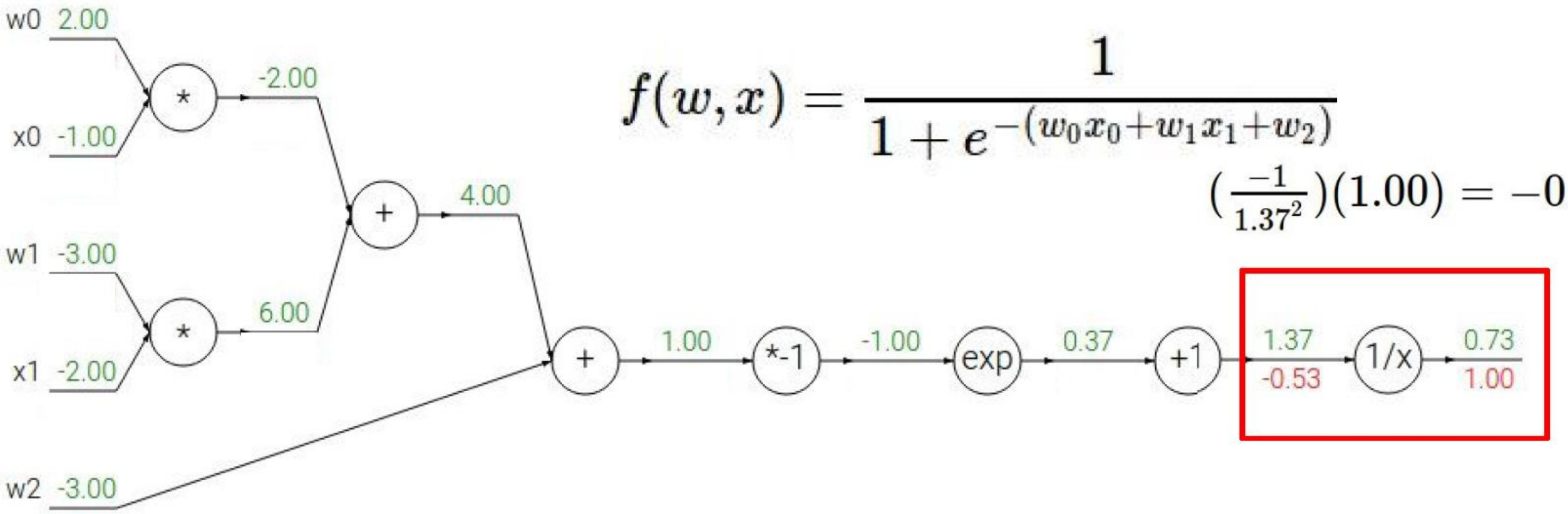
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

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

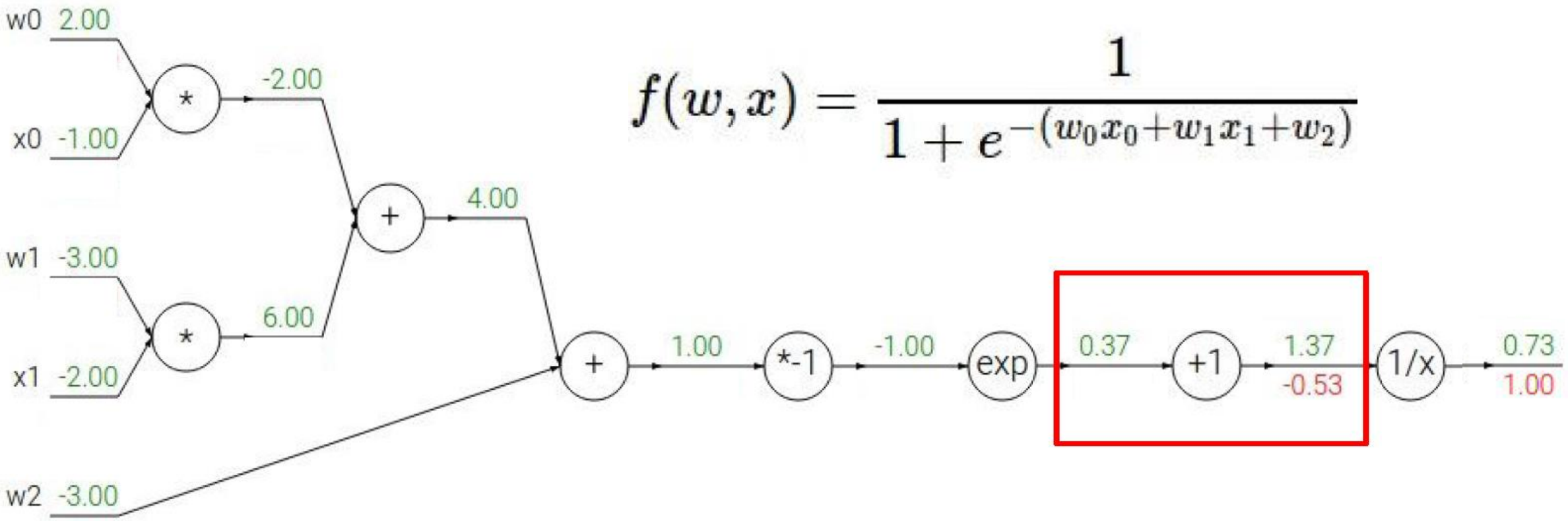
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

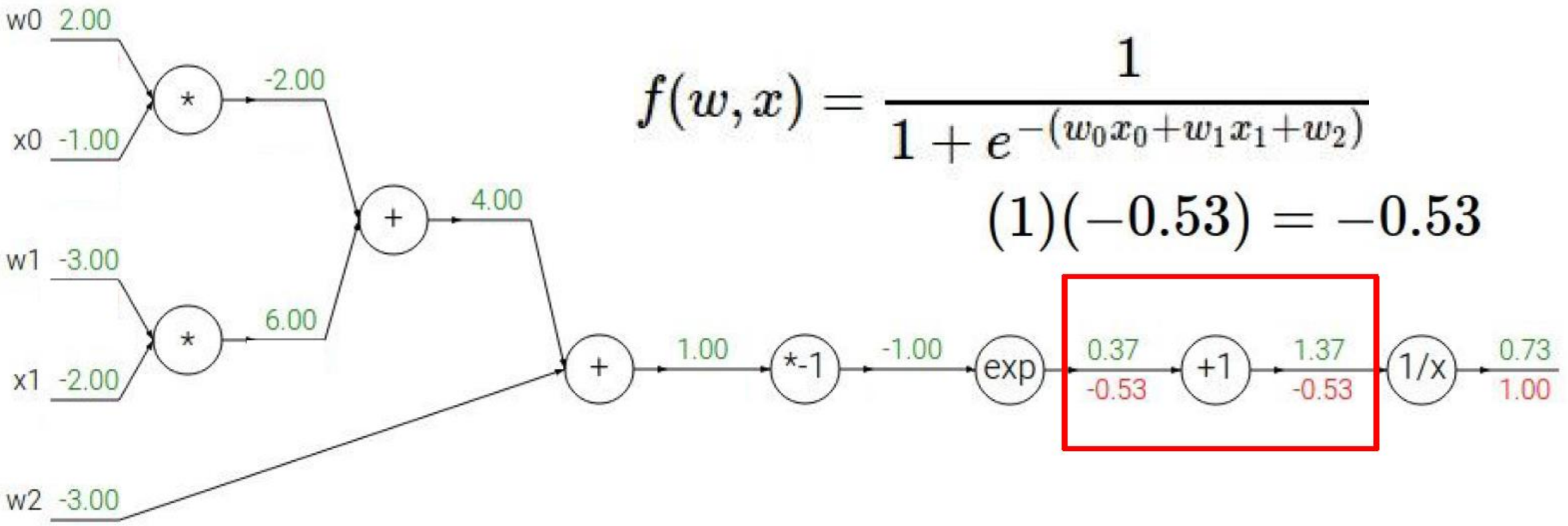
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

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

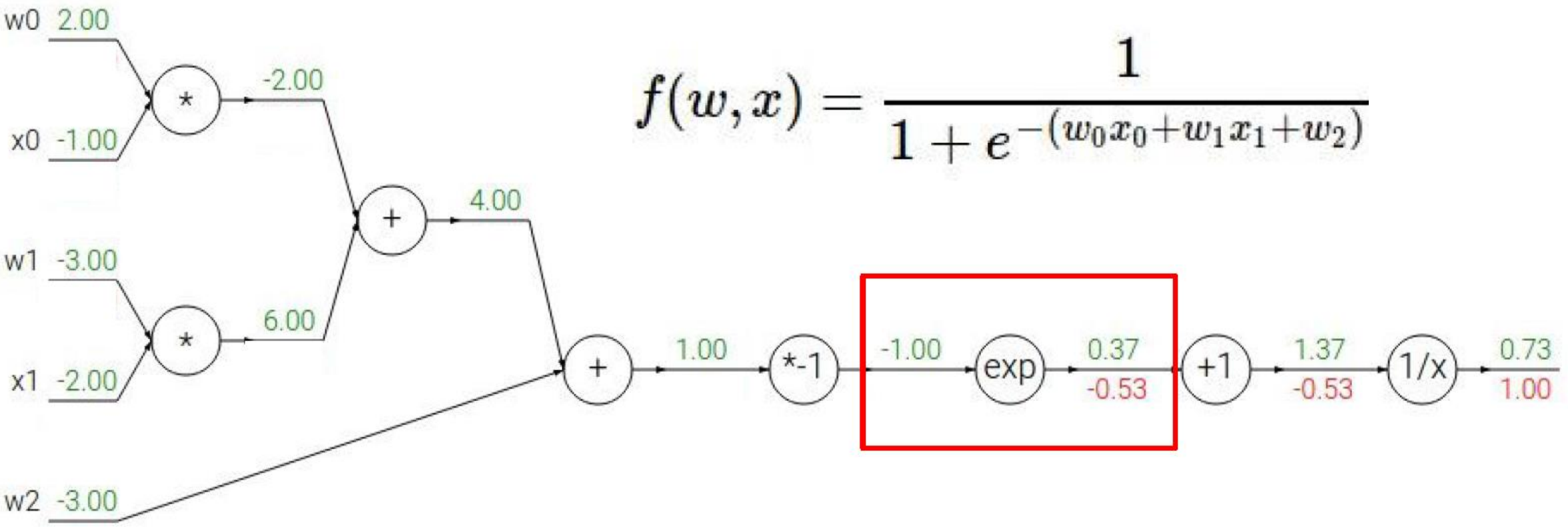
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

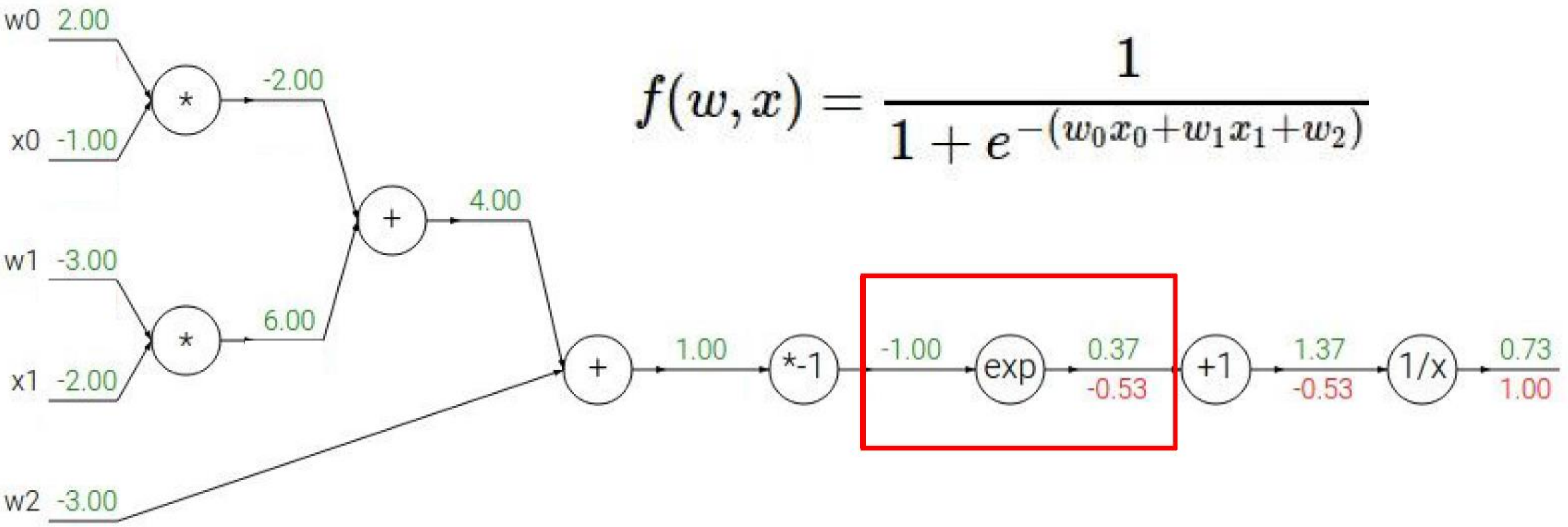
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

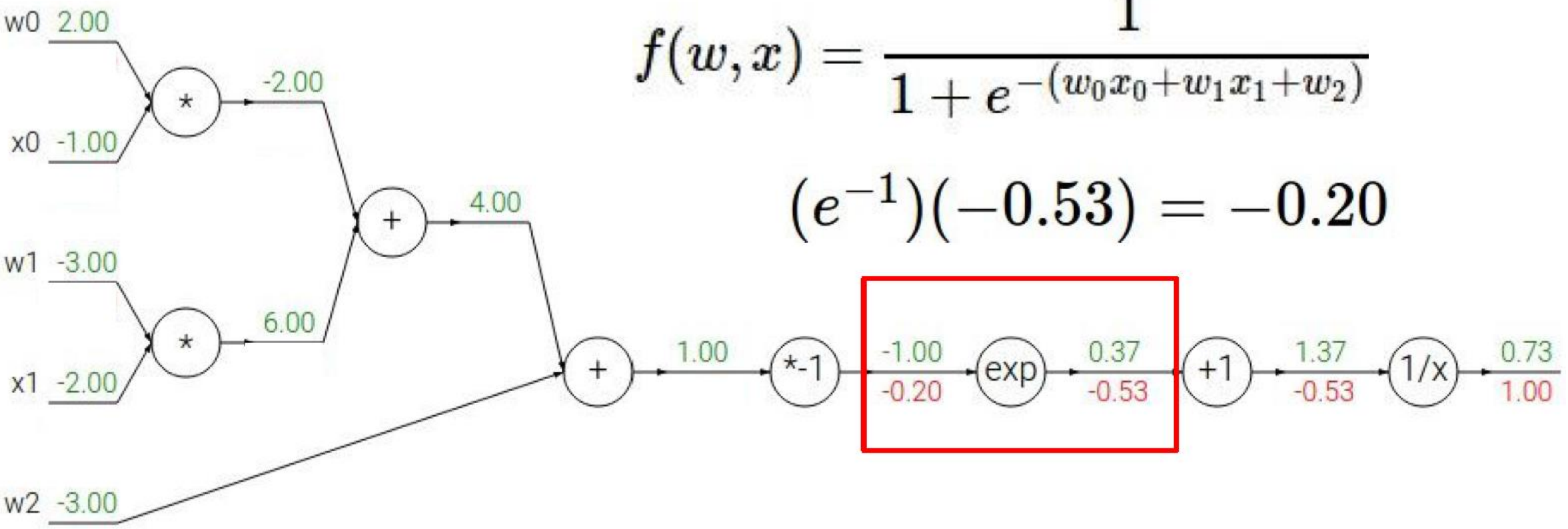
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

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

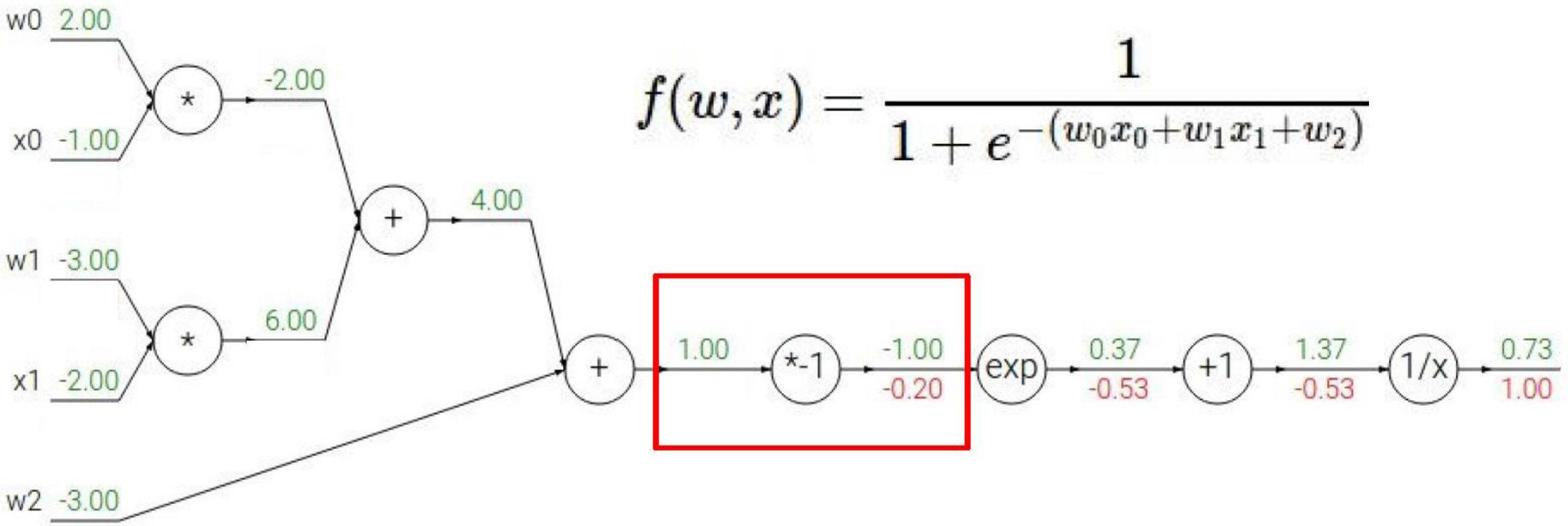
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

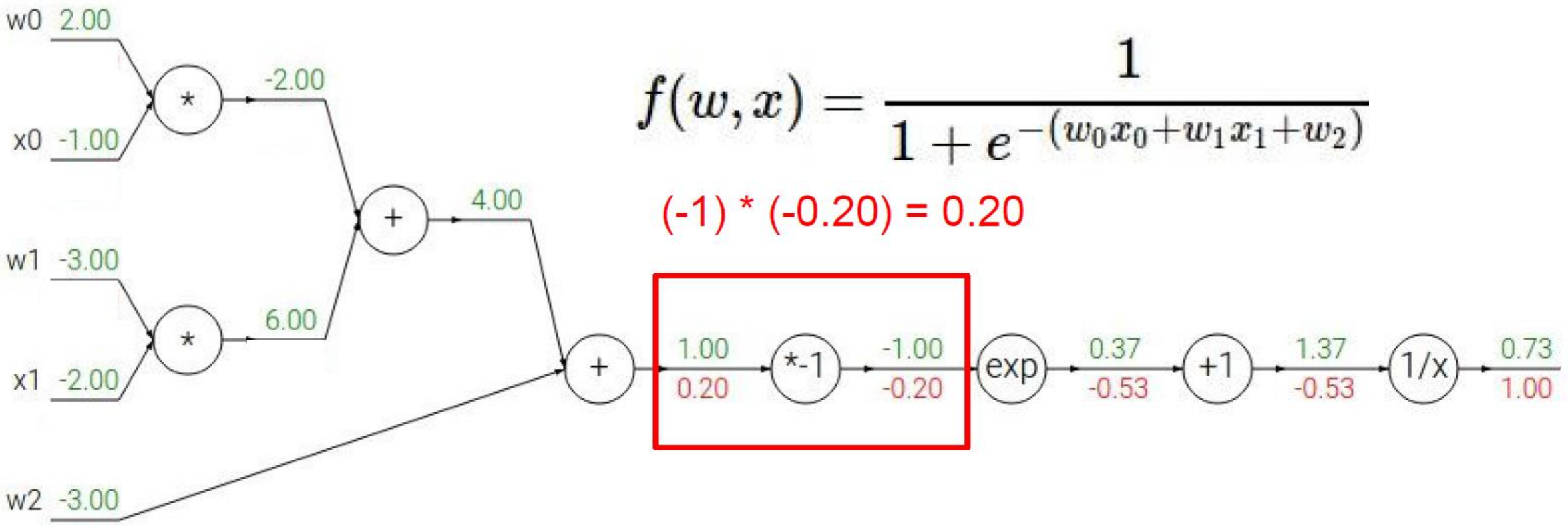
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

(-1) * (-0.20) = 0.20

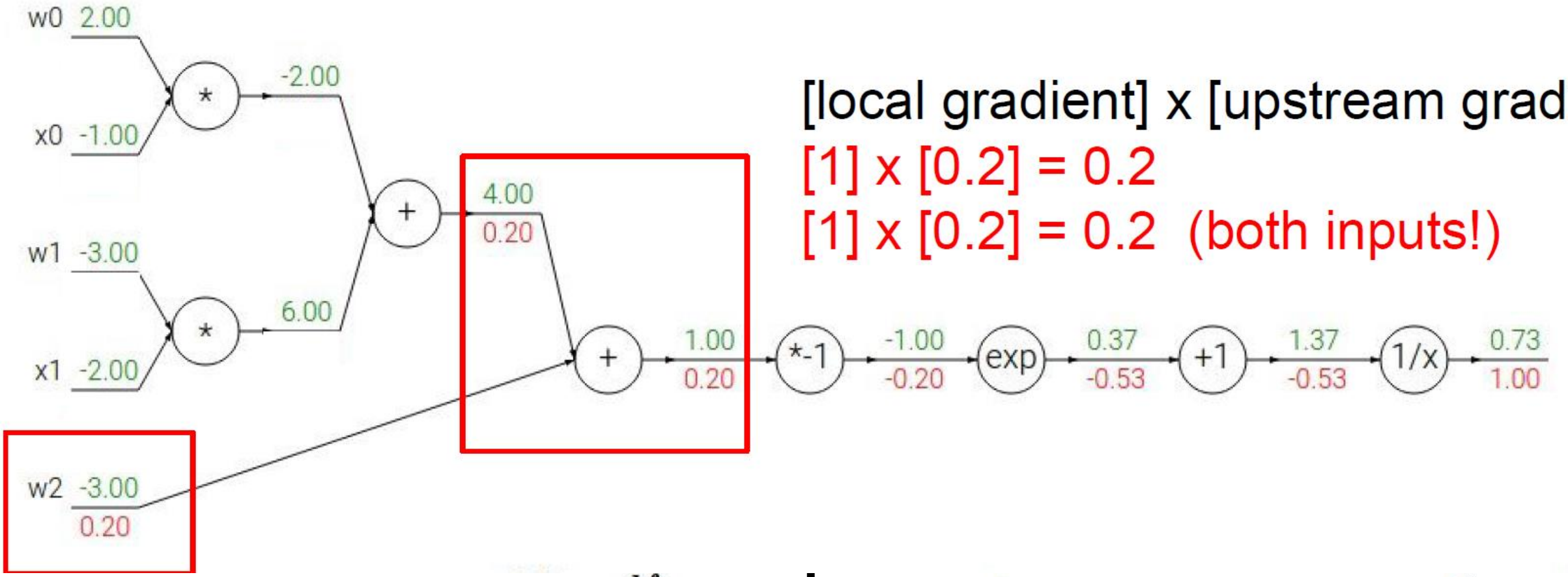
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



[local gradient] x [upstream gradient]

[1] x [0.2] = 0.2

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

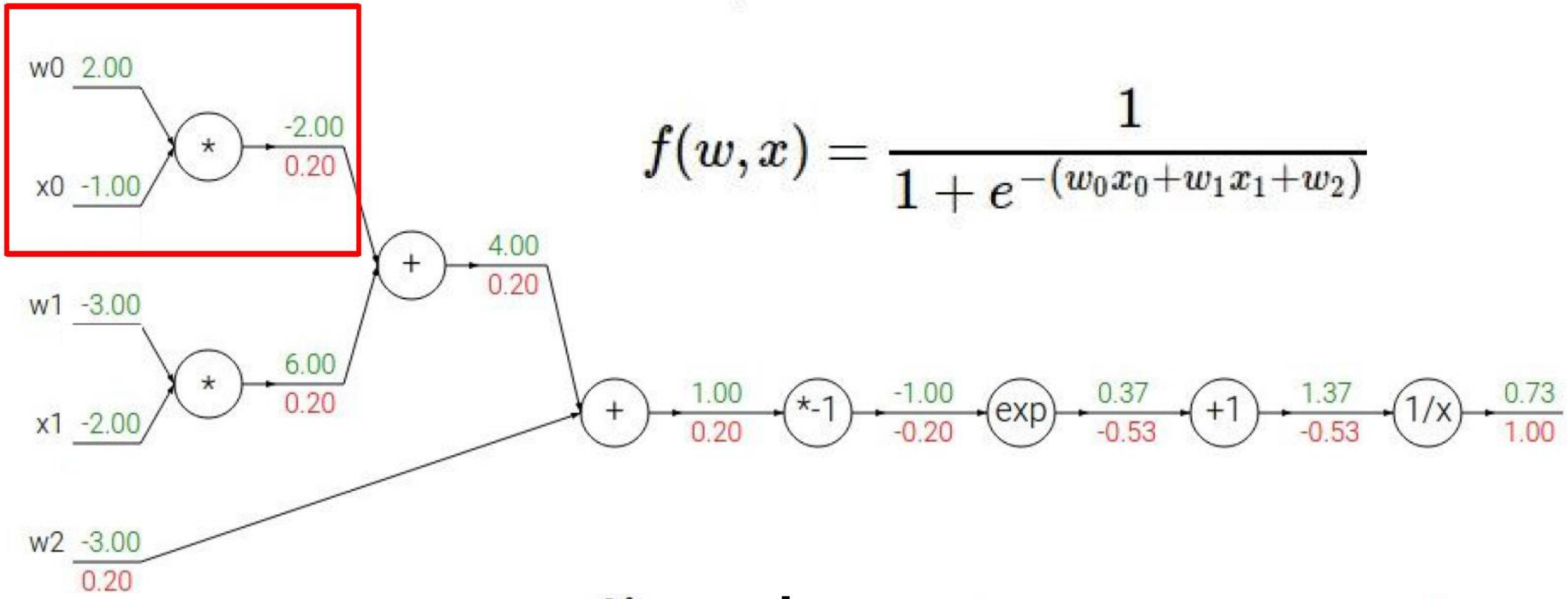
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example



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

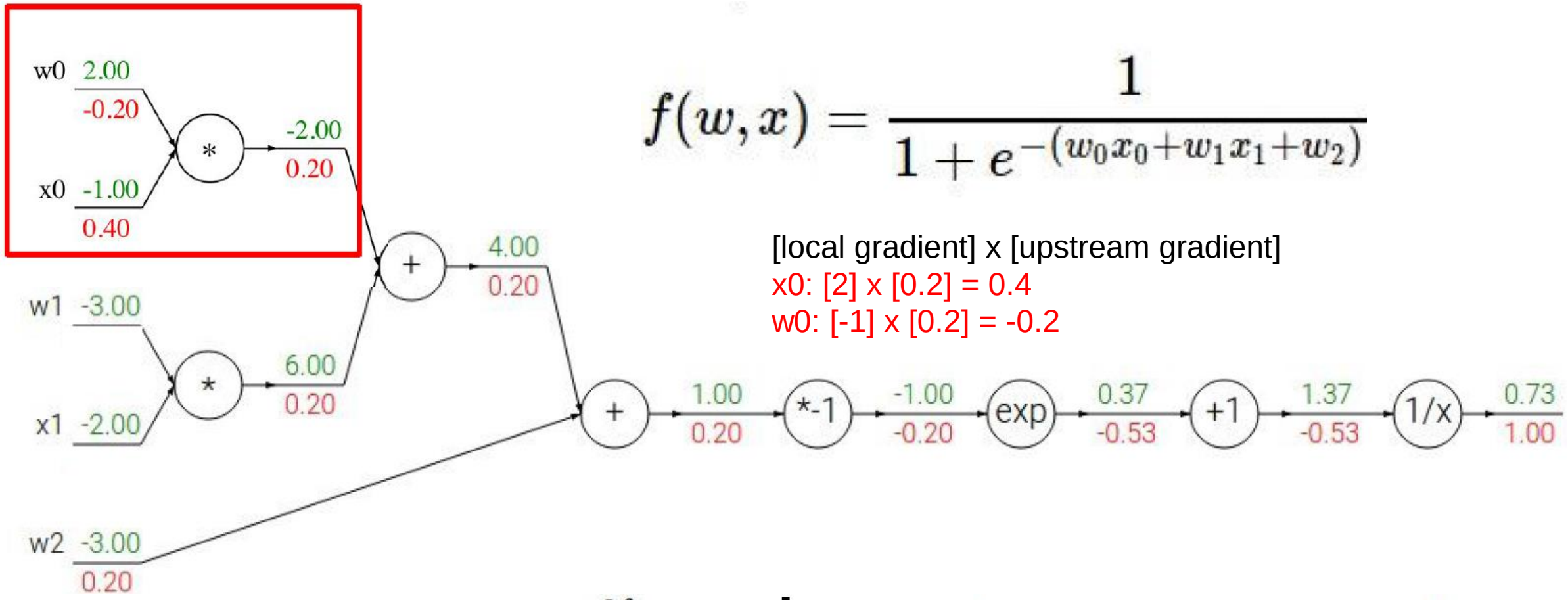
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example



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

[local gradient] x [upstream gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

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

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

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

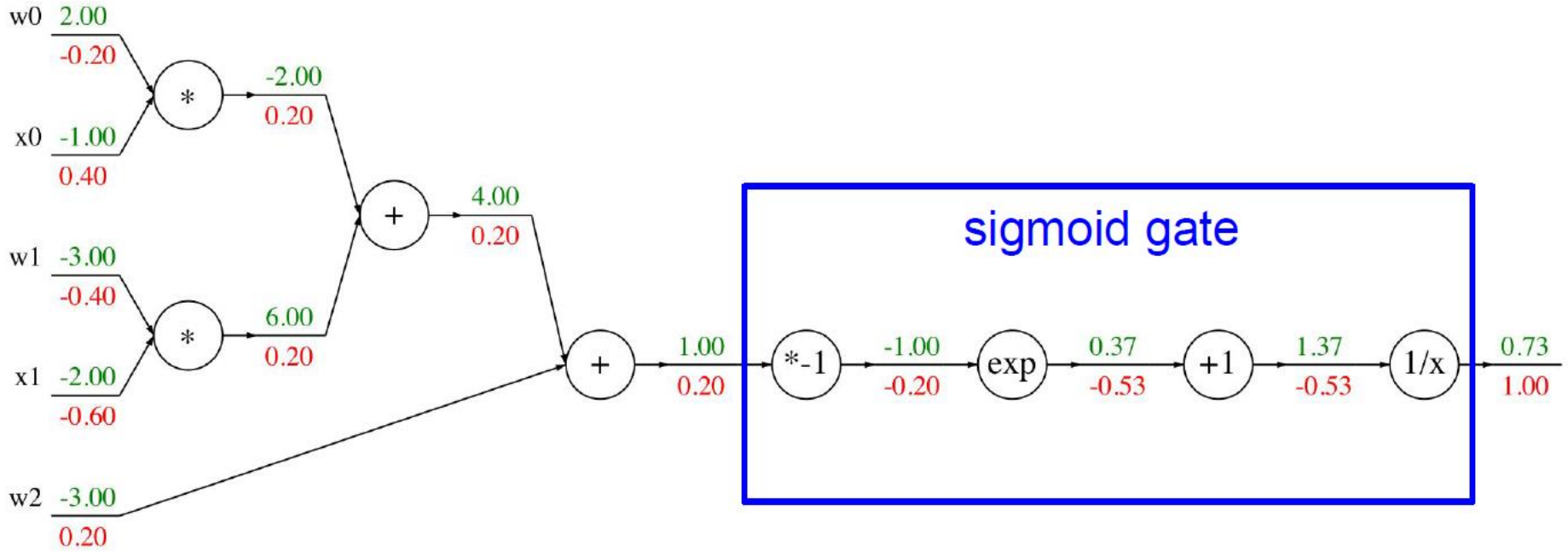
$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Derivative of sigmoid function

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

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

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

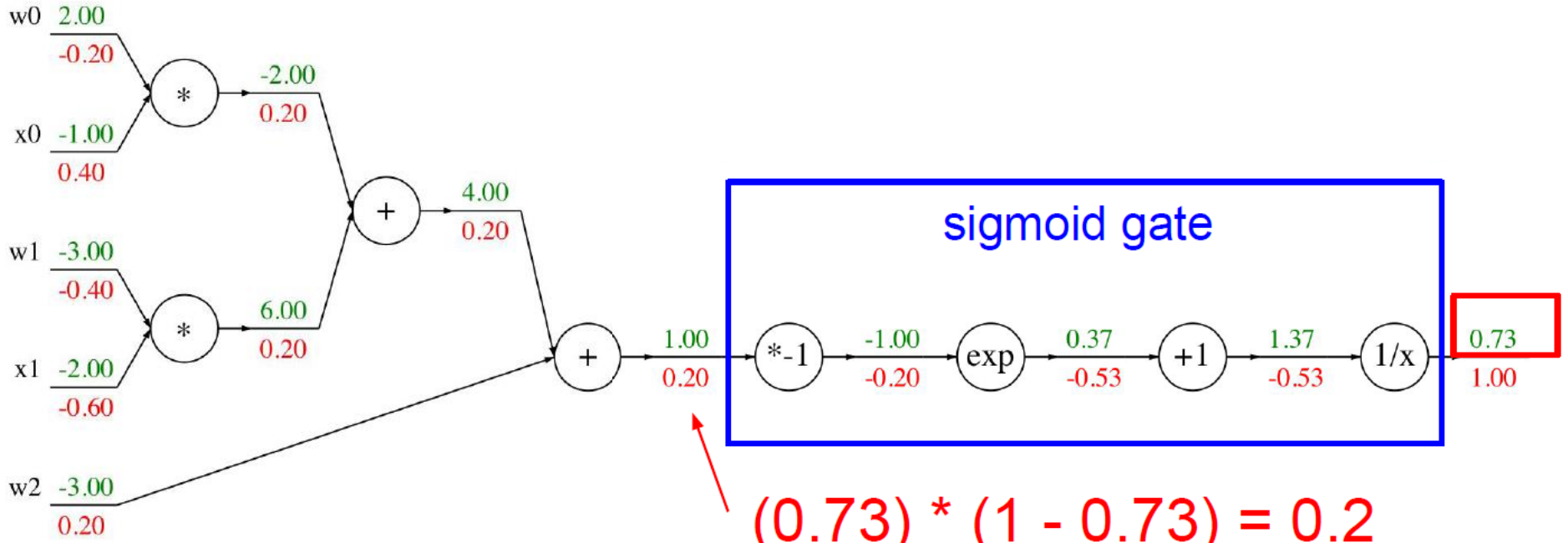


Derivative of sigmoid function

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

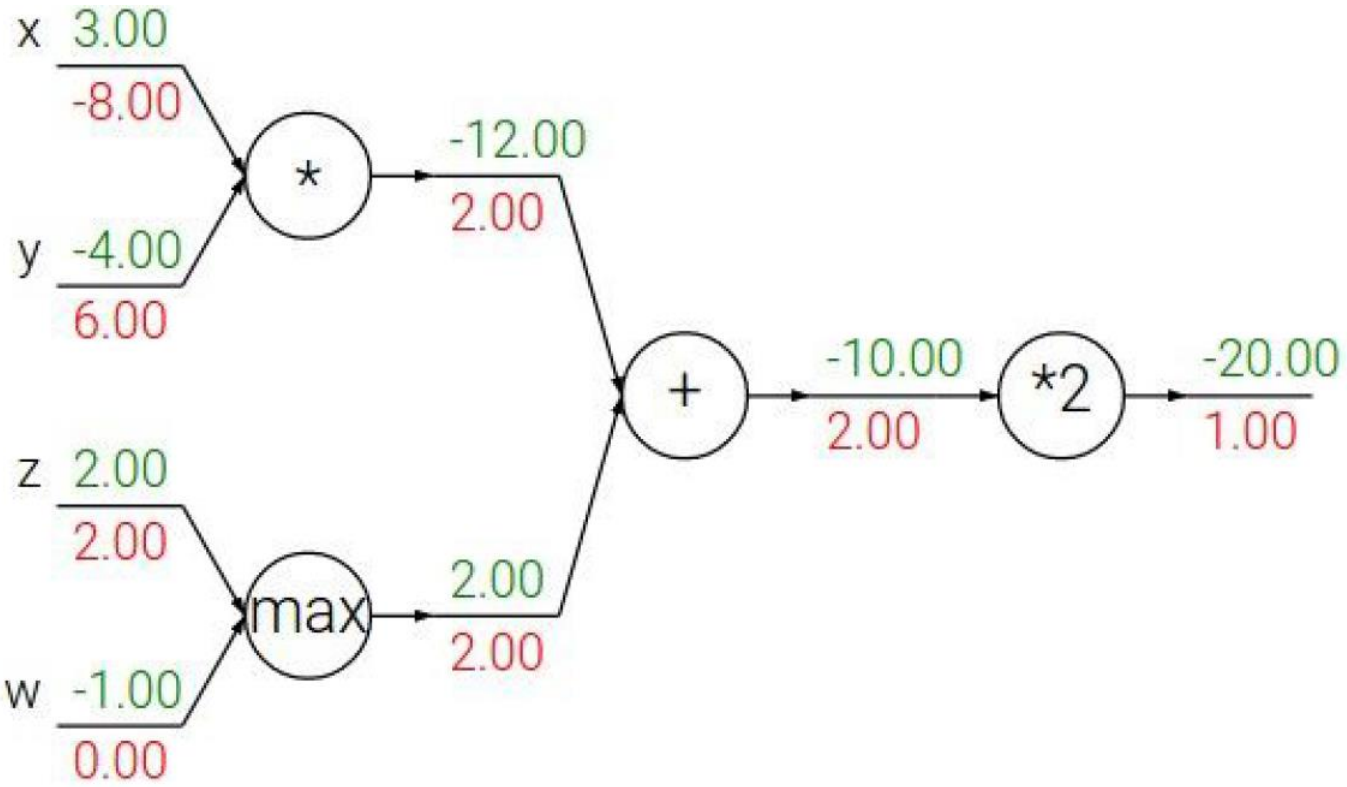
$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$ sigmoid function

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

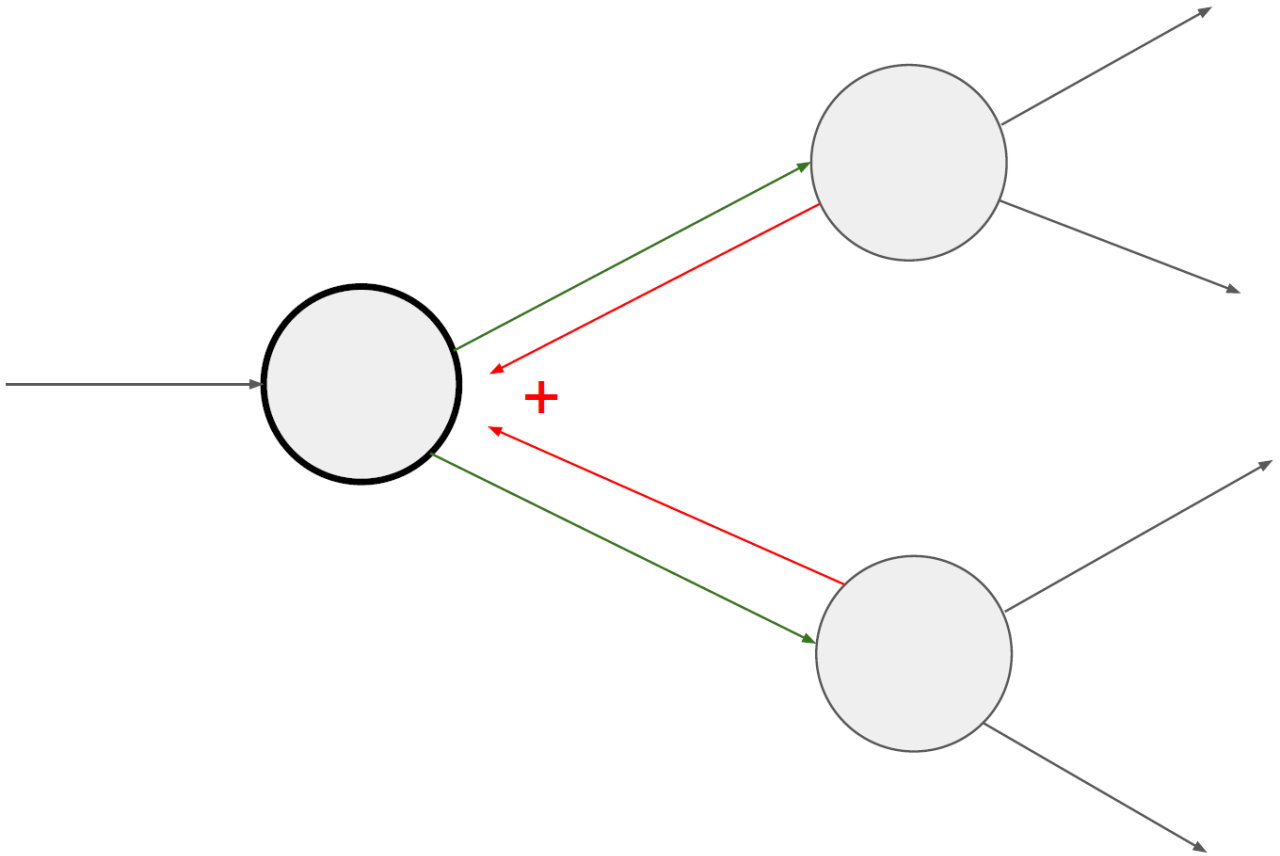


Patterns in backward flow

- **add** gate: gradient distributor
- **max** gate: gradient router

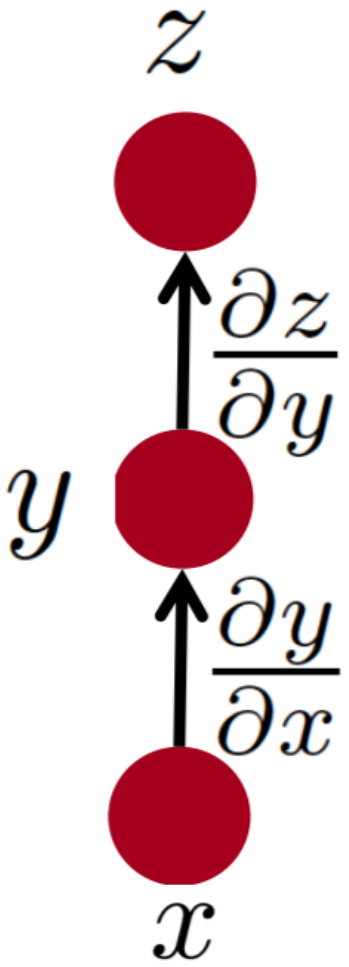


Gradients add at branches



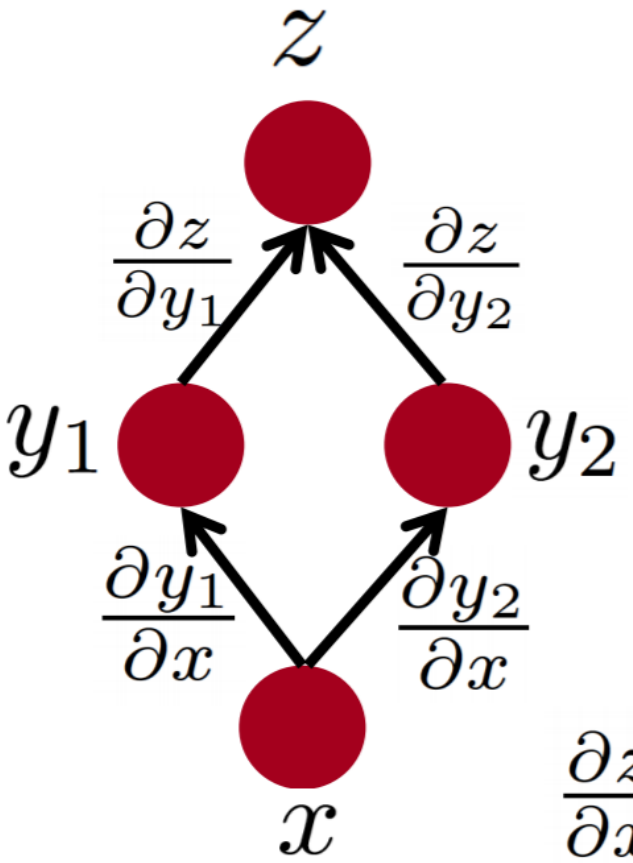
Simple chain rule

- $z = f\big(g(x)\big)$
- $y = g(x)$



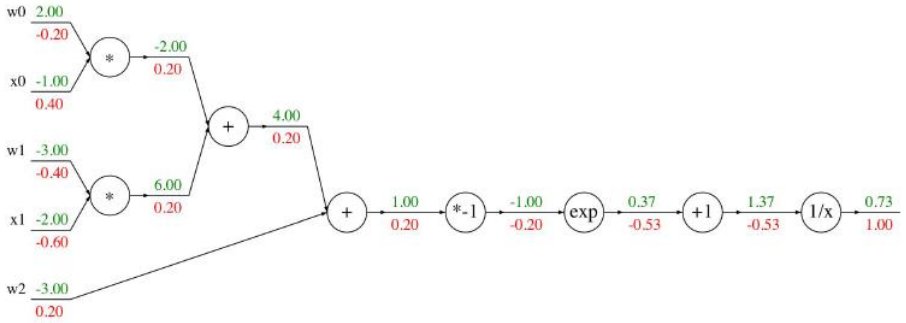
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Multiple paths chain rule



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

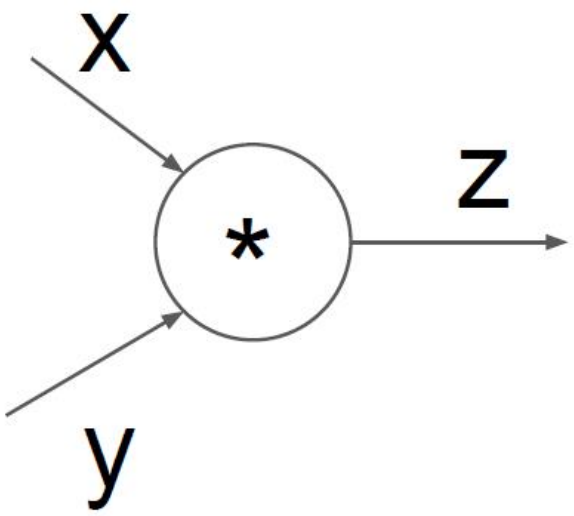
Modularized implementation: forward / backward API



Graph (or Net) object *(rough psuedo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Modularized implementation: forward / backward API



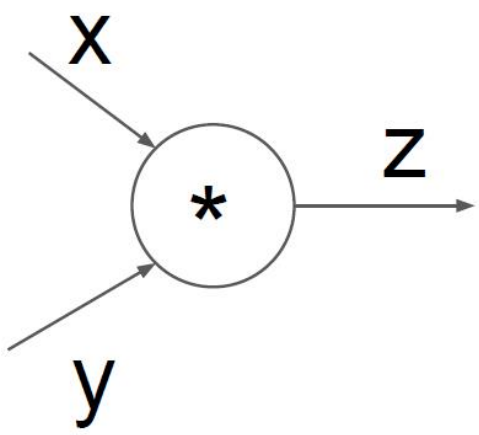
(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

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

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

Modularized implementation: forward / backward API



(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Caffe Layers

Branch: master ▾ | caffe / src / caffe / layers / | Create new file | Upload files | Find file | History

shelhamer committed on GitHub Merge pull request #4630 from BIGene/load_hdf5_fix ... Latest commit e687a71 21 days ago

..

absval_layer.cpp	dismantle layer headers	a year ago
absval_layer.cu	dismantle layer headers	a year ago
accuracy_layer.cpp	dismantle layer headers	a year ago
argmax_layer.cpp	dismantle layer headers	a year ago
base_conv_layer.cpp	enable dilated deconvolution	a year ago
base_data_layer.cpp	Using default from proto for prefetch	3 months ago
base_data_layer.cu	Switched multi-GPU to NCCL	3 months ago
batch_norm_layer.cpp	Add missing spaces besides equal signs in batch_norm_layer.cpp	4 months ago
batch_norm_layer.cu	dismantle layer headers	a year ago
batch_reindex_layer.cpp	dismantle layer headers	a year ago
batch_reindex_layer.cu	dismantle layer headers	a year ago
bias_layer.cpp	Remove incorrect cast of gemm int arg to Dtype in BiasLayer	a year ago
bias_layer.cu	Separation and generalization of ChannelwiseAffineLayer into BiasLayer	a year ago
bnll_layer.cpp	dismantle layer headers	a year ago
bnll_layer.cu	dismantle layer headers	a year ago
concat_layer.cpp	dismantle layer headers	a year ago
concat_layer.cu	dismantle layer headers	a year ago
contrastive_loss_layer.cpp	dismantle layer headers	a year ago
contrastive_loss_layer.cu	dismantle layer headers	a year ago
conv_layer.cpp	add support for 2D dilated convolution	a year ago
conv_layer.cu	dismantle layer headers	a year ago
crop_layer.cpp	remove redundant operations in Crop layer (#5138)	2 months ago
crop_layer.cu	remove redundant operations in Crop layer (#5138)	2 months ago
cudnn_conv_layer.cpp	dismantle layer headers	a year ago
cudnn_conv_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_lcn_layer.cpp	dismantle layer headers	a year ago
cudnn_lcn_layer.cu	dismantle layer headers	a year ago
cudnn_lrn_layer.cpp	dismantle layer headers	a year ago
cudnn_lrn_layer.cu	dismantle layer headers	a year ago
cudnn_pooling_layer.cpp	dismantle layer headers	a year ago
cudnn_pooling_layer.cu	dismantle layer headers	a year ago
cudnn_relu_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_relu_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_softmax_layer.cpp	dismantle layer headers	a year ago
cudnn_softmax_layer.cu	dismantle layer headers	a year ago
cudnn_tanh_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_tanh_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
data_layer.cpp	Switched multi-GPU to NCCL	3 months ago
deconv_layer.cpp	enable dilated deconvolution	a year ago
deconv_layer.cu	dismantle layer headers	a year ago
dropout_layer.cpp	supporting N-D Blobs in Dropout layer Reshape	a year ago
dropout_layer.cu	dismantle layer headers	a year ago
dummy_data_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cu	dismantle layer headers	a year ago
elu_layer.cpp	ELU layer with basic tests	a year ago
elu_layer.cu	ELU layer with basic tests	a year ago
embed_layer.cpp	dismantle layer headers	a year ago
embed_layer.cu	dismantle layer headers	a year ago
euclidean_loss_layer.cpp	dismantle layer headers	a year ago
euclidean_loss_layer.cu	dismantle layer headers	a year ago
exp_layer.cpp	Solving issue with exp layer with base e	a year ago
exp_layer.cu	dismantle layer headers	a year ago

Caffe sigmoid layer

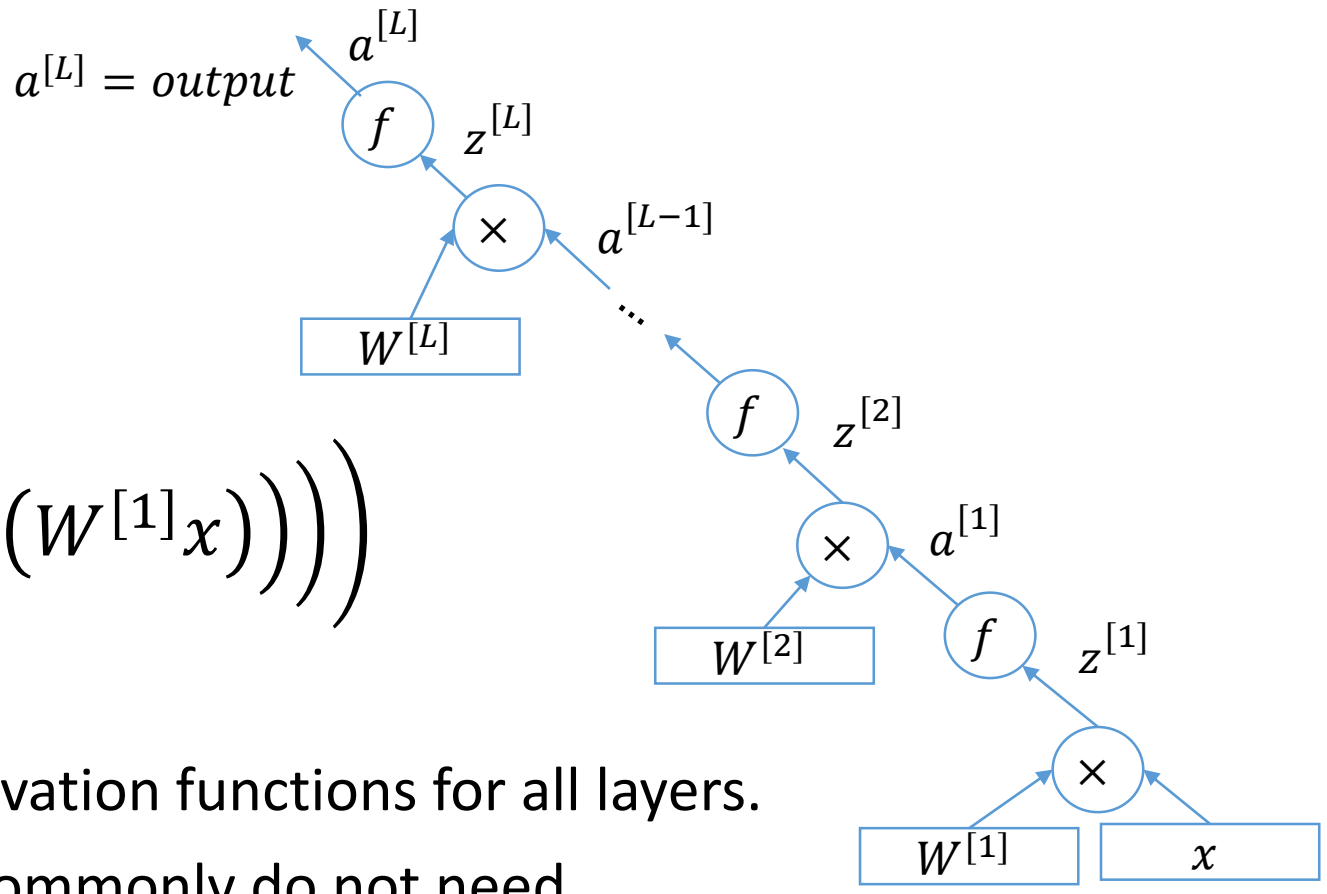
```cpp
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```

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

$$(1-\sigma(x))\,\sigma(x)$$ * top_diff (chain rule)

Output as a composite function

$$Output = a^{[L]}$$
$$= f\left(z^{[L]}\right)$$
$$= f\left(W^{[L]}a^{[L-1]}\right)$$
$$= f\left(W^{[L]}f(W^{[L-1]}a^{[L-2]}\right)$$
$$= f\left(W^{[L]}f\left(W^{[L-1]}\ldots f\left(W^{[2]}f\left(W^{[1]}x\right)\right)\right)\right)$$

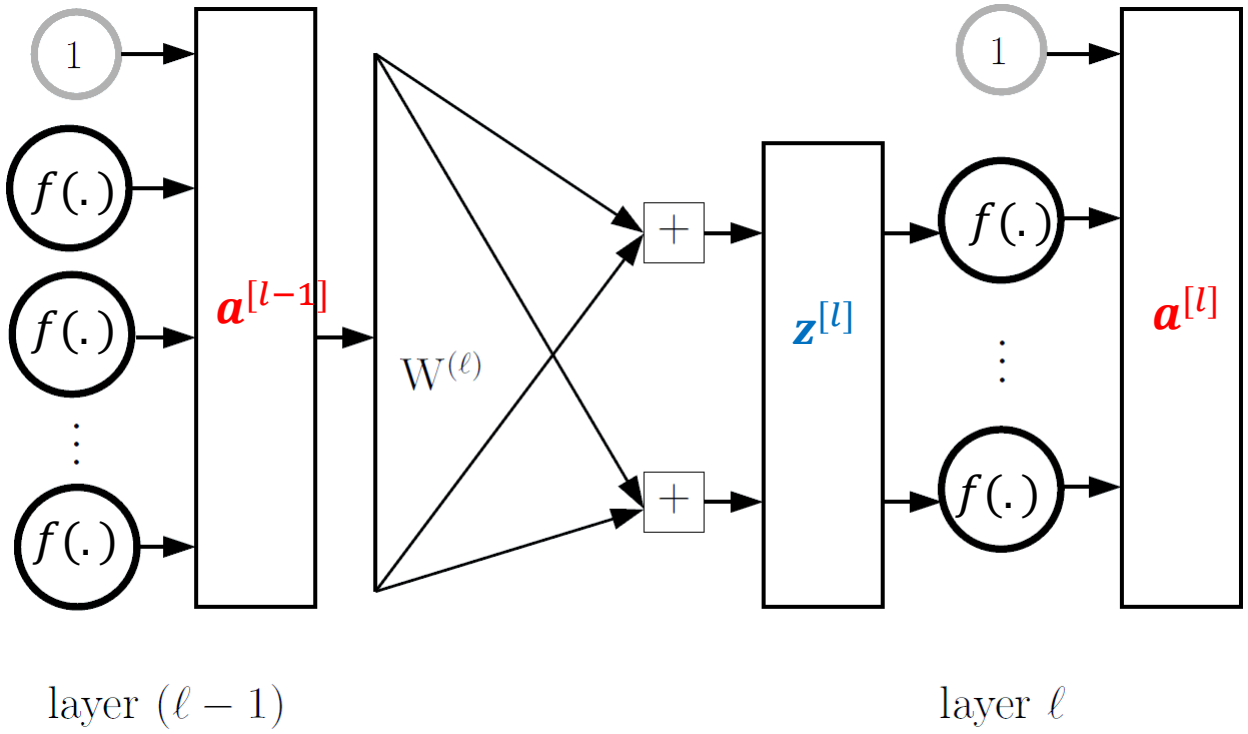


For convenience, we use the same activation functions for all layers.

However, output layer neurons most commonly do not need activation function (they show class scores or real-valued targets.)

Backpropagation: Notation

- $\boldsymbol{a}^{[0]} \leftarrow Input$
- $output \leftarrow \boldsymbol{a}^{[L]}$



Backpropagation: Last layer gradient

$$\frac{\partial E_n}{\partial W_{ij}^{[L]}} = \frac{\partial E_n}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial W_{ij}^{[L]}}$$

For squared error loss:

$$E_n = \left(a^{[L]} - y^{(n)}\right)^2$$

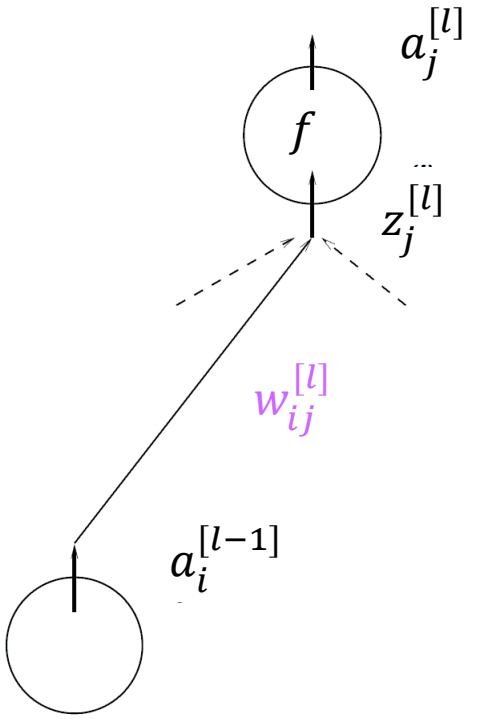
$$\frac{\partial E}{\partial a^{[L]}} = 2(y - a^{[L]})$$

$$\frac{\partial a^{[L]}}{\partial W_{ij}^{[L]}} = f'\left(z_j^{[L]}\right) \frac{\partial z_j^{[L]}}{\partial W_{ij}^{[L]}}$$

$$= f'\left(z_j^{[L]}\right) a_i^{[L-1]}$$

$$a_i^{[l]} = f\left(z_i^{[l]}\right)$$

$$z_j^{[l]} = \sum_{j=0}^{M} w_{ij}^{[l]} a_i^{[l-1]}$$



Backpropagation:

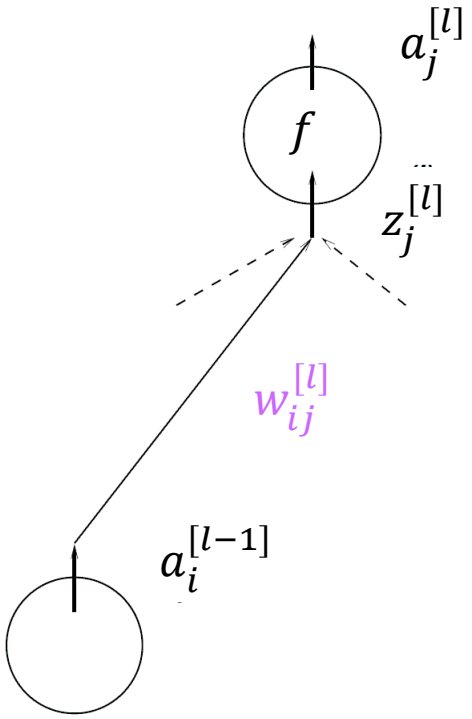
$$\frac{\partial E_n}{\partial w_{ij}^{[l]}} = \frac{\partial E_n}{\partial a_j^{[l]}} \times \frac{\partial a_j^{[l]}}{\partial w_{ij}^{[l]}}$$

$$= \delta_j^{[l]} \times a_i^{[l-1]} \times f'\left(z_i^{[l]}\right)$$

$$a_i^{[l]} = f\left(z_i^{[l]}\right)$$

$$z_j^{[l]} = \sum_{j=0}^{M} w_{ij}^{[l]} a_i^{[l-1]}$$

- $\delta_j^{[l]} = \frac{\partial E_n}{\partial a_j^{[l]}}$ is the sensitivity of the output to $a_j^{[l]}$
- sensitivity vectors can be obtained by running a backward process in the network architecture (hence the name backpropagation.)



$\delta_i^{[l-1]}$ from $\delta_i^{[l]}$

We will compute $\boldsymbol{\delta}^{[l-1]}$ from $\boldsymbol{\delta}^{[l]}$:

$$\delta_i^{[l-1]} = \frac{\partial E_n}{\partial a_i^{[l-1]}}$$

$$= \sum_{j=1}^{d^{[l]}} \frac{\partial E_n}{\partial a_j^{[l]}} \times \frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} \times \frac{\partial z_j^{[l]}}{\partial a_i^{[l-1]}}$$

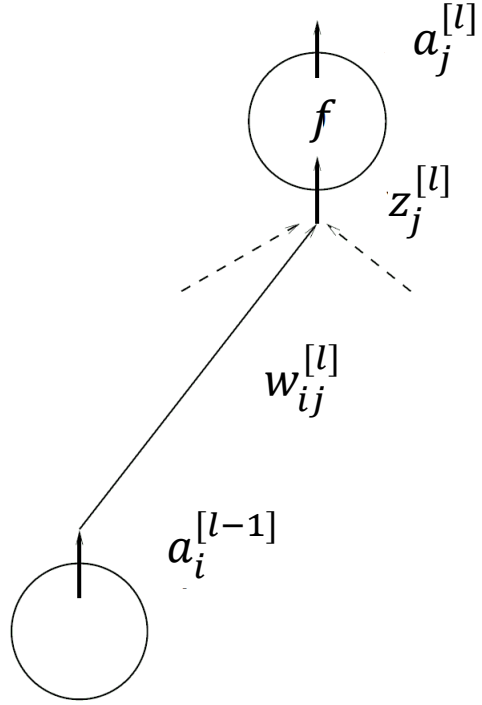
$$= \sum_{j=1}^{d^{[l]}} \frac{\partial E_n}{\partial a_j^{[l]}} \times f'\left(z_i^{[l]}\right) \times w_{ij}^{[l]}$$

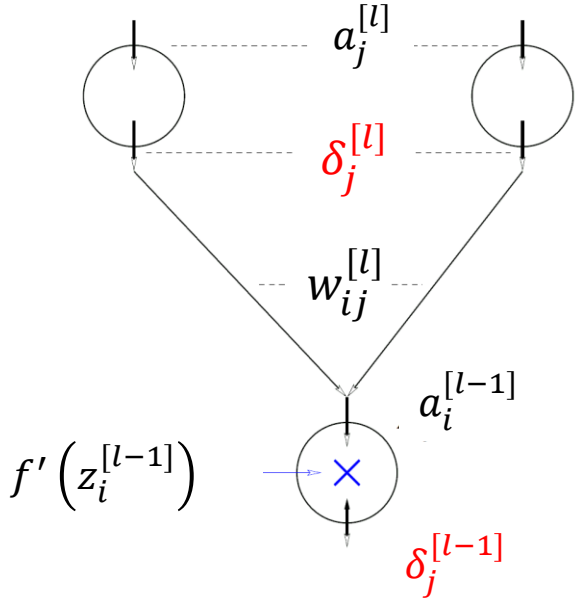
$$= \sum_{j=1}^{d^{[l]}} \delta_j^{[l]} \times f'\left(z_i^{[l]}\right) \times w_{ij}^{[l]}$$

$$= f'\left(z_i^{[l]}\right) \times \sum_{j=1}^{d^{[l]}} \delta_j^{[l]} \times w_{ij}^{[l]}$$

$$a_i^{[l-1]} = f\left(z_i^{[l-1]}\right)$$

$$z_j^{[l]} = \sum_{j=0}^{M} w_{ij}^{[l]} a_i^{[l-1]}$$





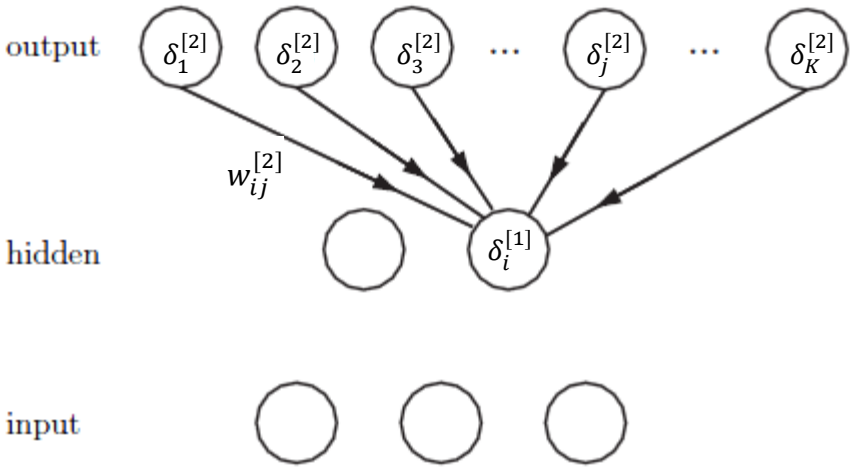
Find and save $\boldsymbol{\delta}^{[L]}$

- Called error, computed recursively in backward manner

- For the final layer $l = L$:

$$\delta_j^{[L]} = \frac{\partial E_n}{\partial a_j^{[L]}}$$

Backpropagation of Errors

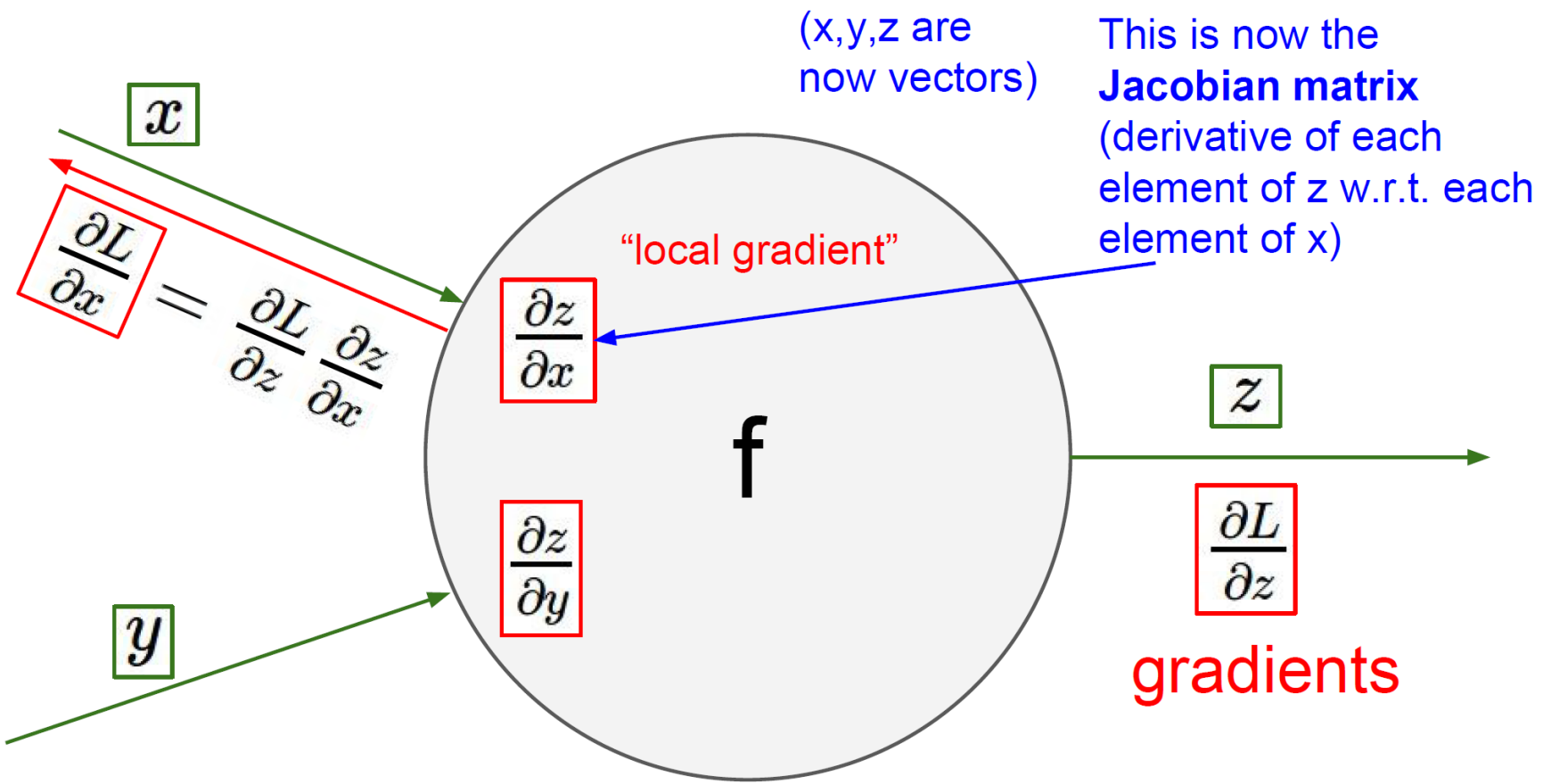


$$E_n = \sum_j \left(a_j^{[L]} - y_j^{(n)}\right)^2$$

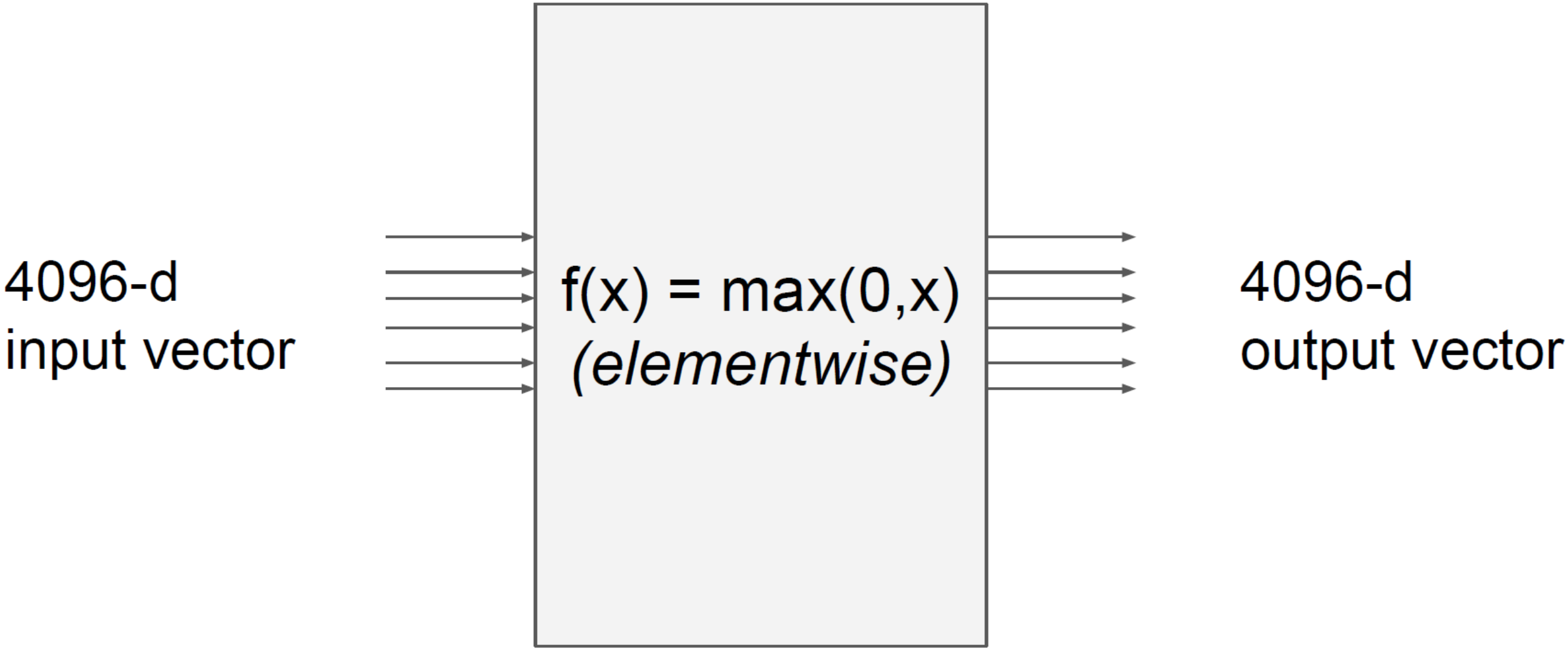
$$\delta_j^{[2]} = 2\left(a_j^{[2]} - y_j^{(n)}\right) f'\left(z_j^{[2]}\right)$$

$$\delta_i^{[1]} = f'\left(z_i^{[1]}\right) \times \sum_{j=1}^{d^{[2]}} \delta_j^{[2]} \times w_{ij}^{[2]}$$

Gradients for vectorized code



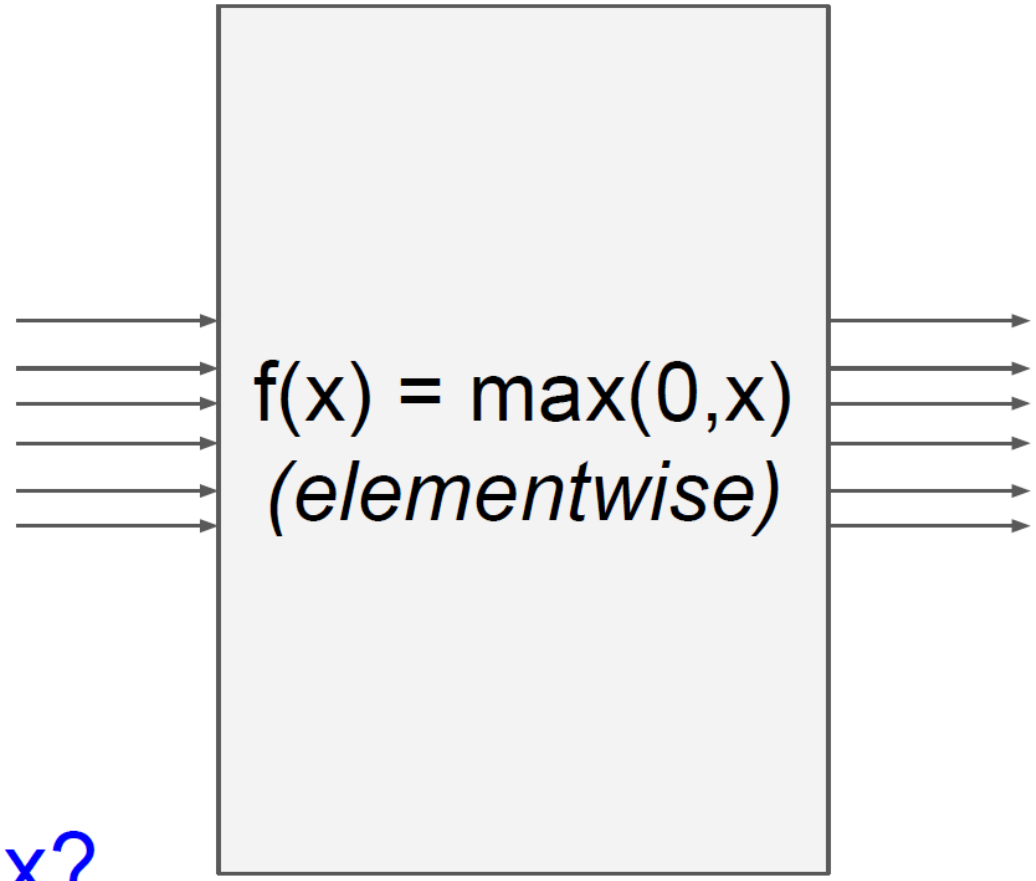
Vectorized operations



Vectorized operations

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

Jacobian matrix

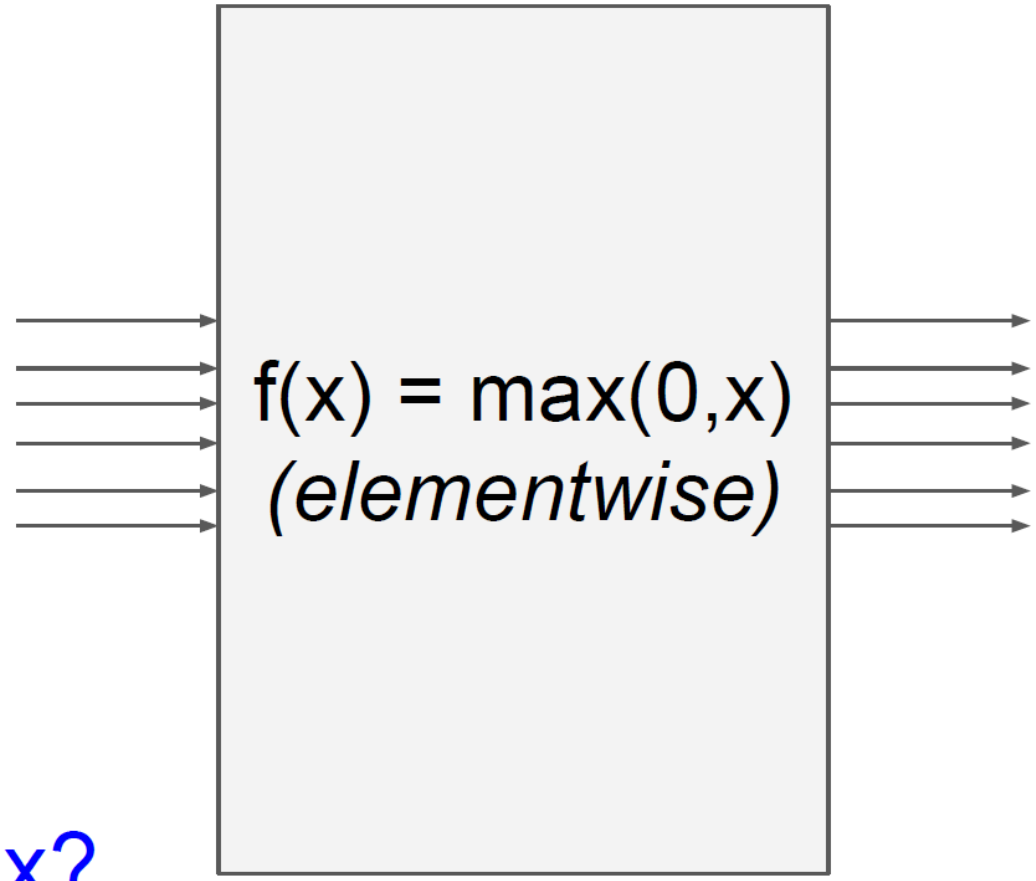


Q: what is the size of the Jacobian matrix?

Vectorized operations

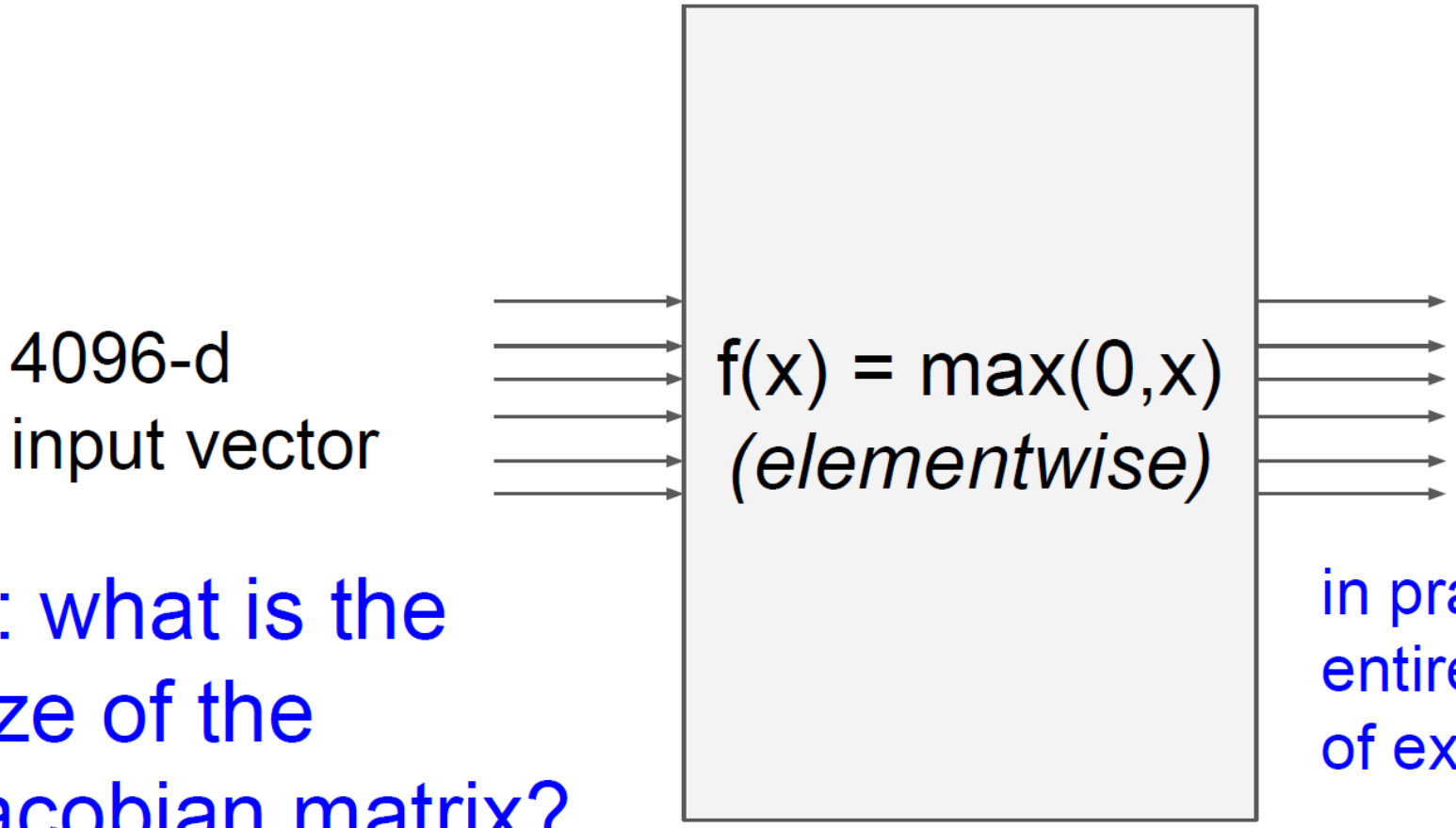
$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Jacobian matrix



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

Vectorized operations



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

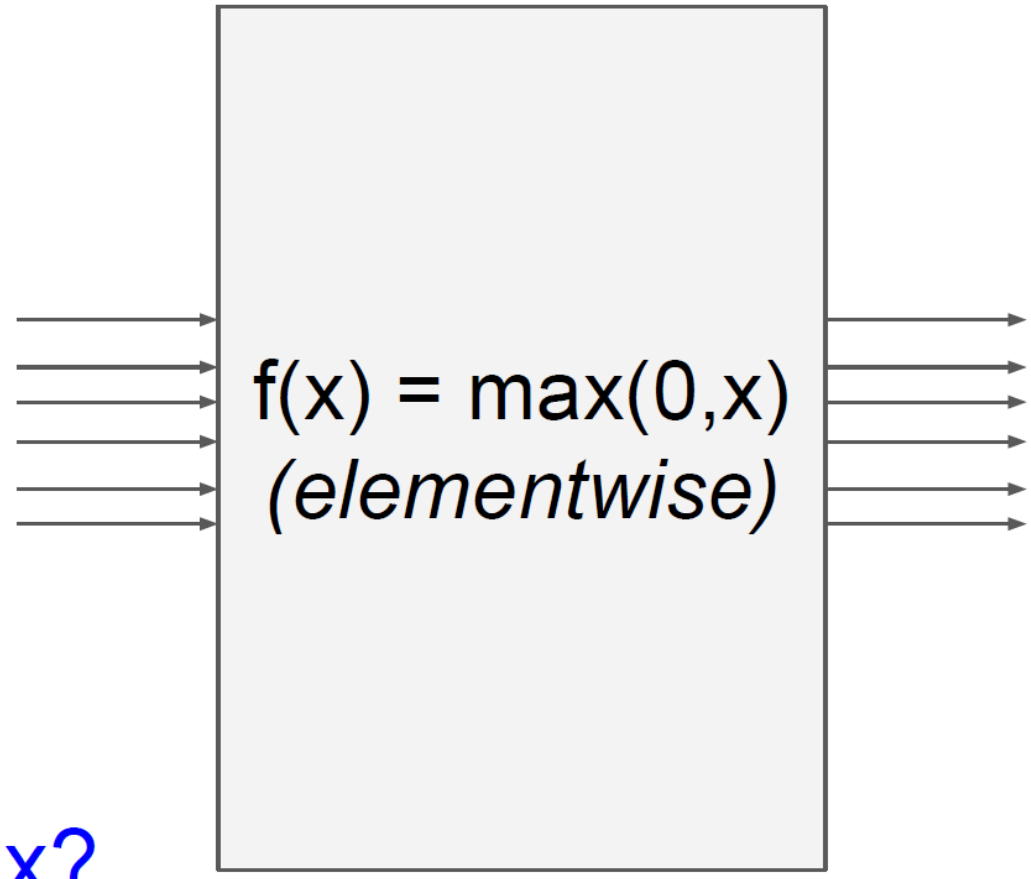
in practice we process an entire minibatch (e.g. 100) of examples at one time:

i.e. Jacobian would technically be a [409,600 x 409,600] matrix :\

Vectorized operations

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

Jacobian matrix



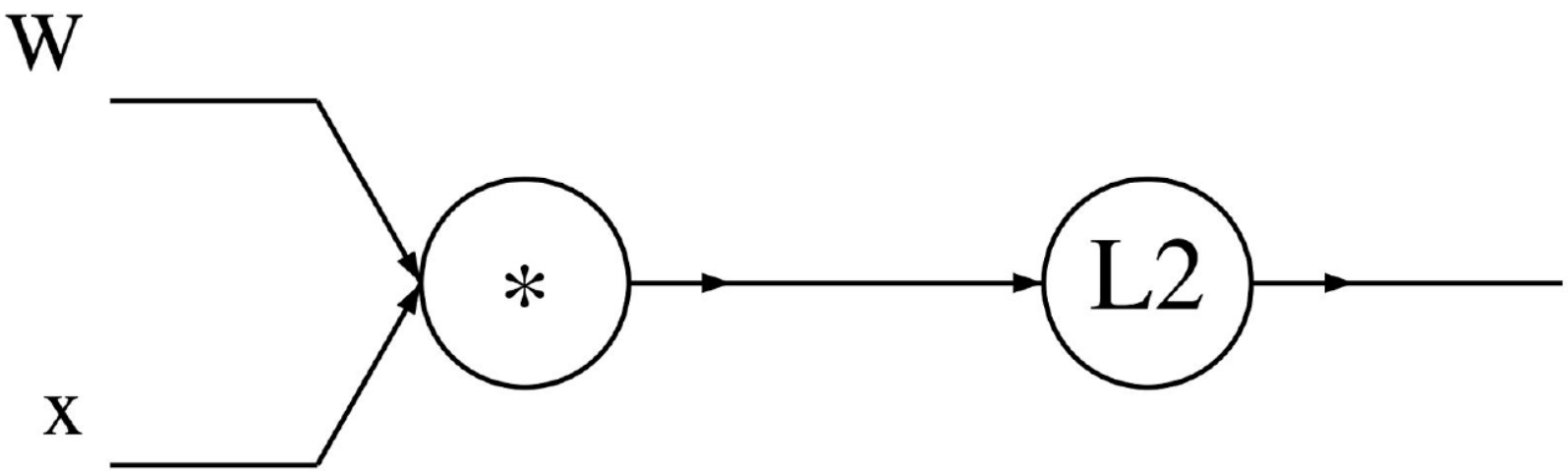
Q: what is the size of the Jacobian matrix? [4096 x 4096!]

Q2: what does it look like?

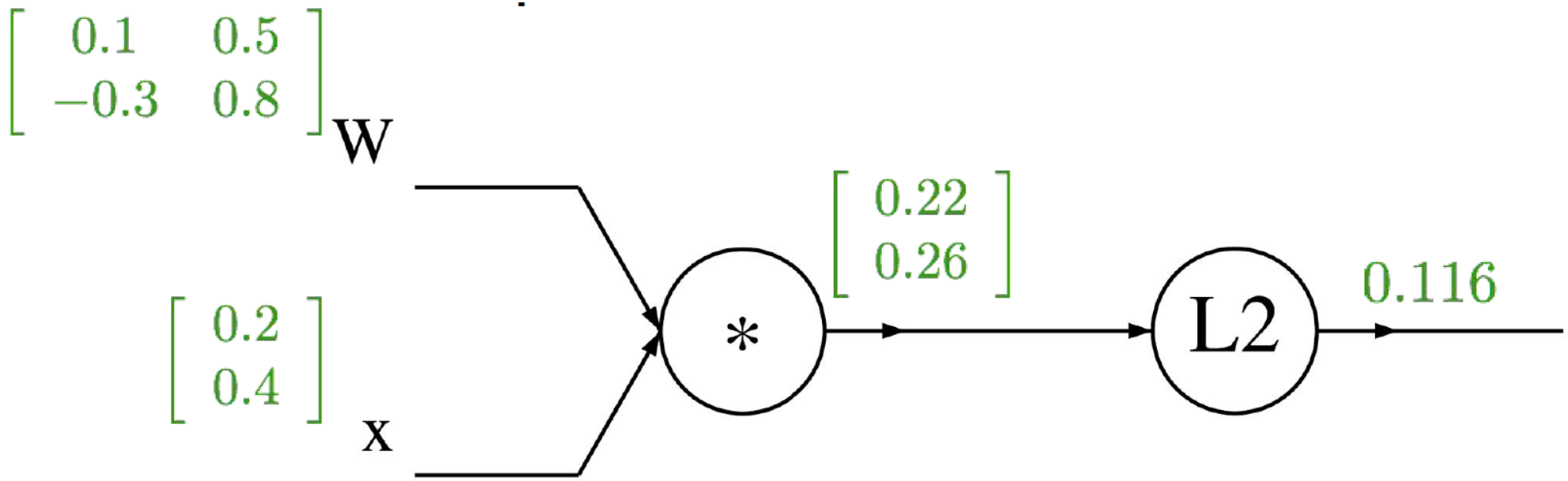
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n$ $\in \mathbb{R}^{n \times n}$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



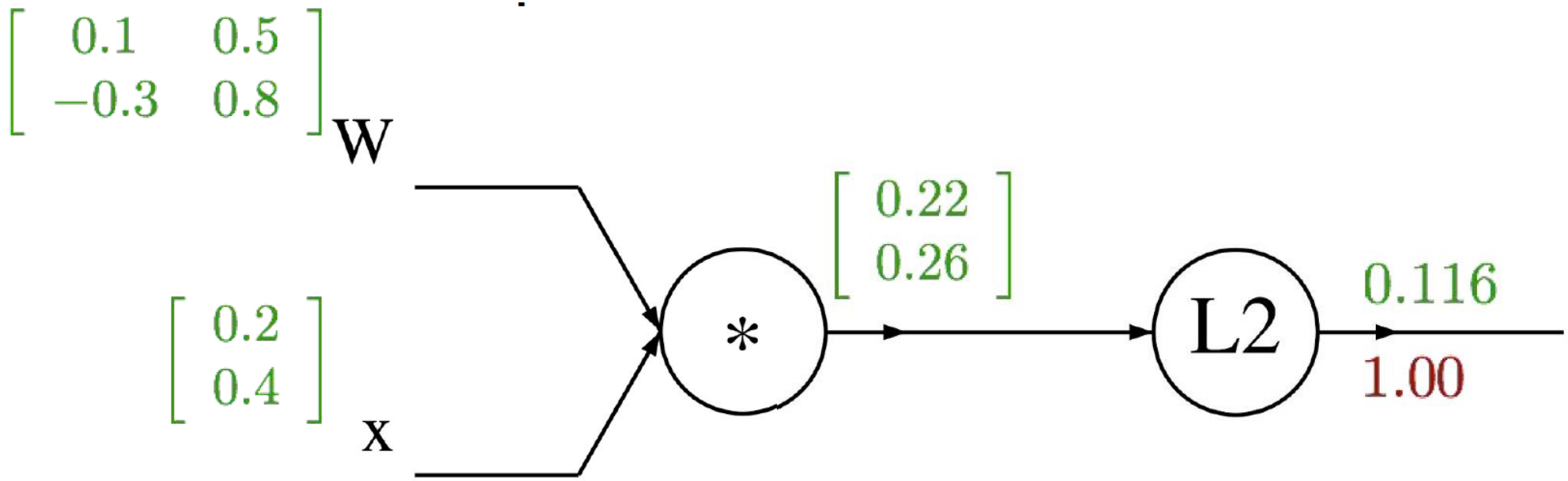
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

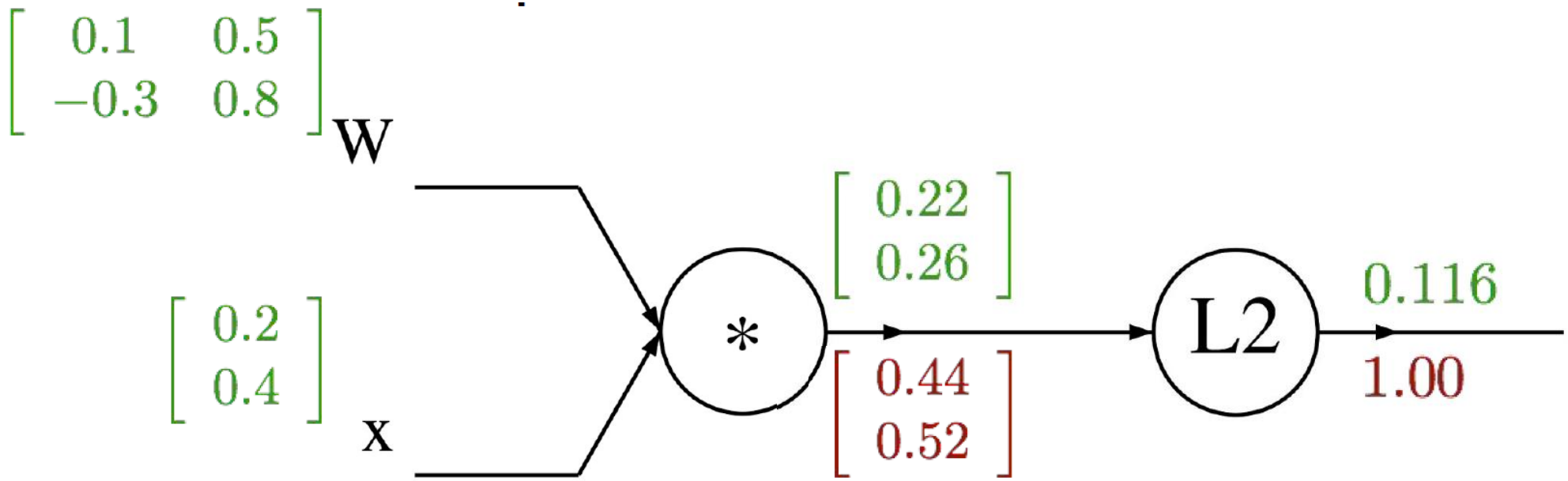
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



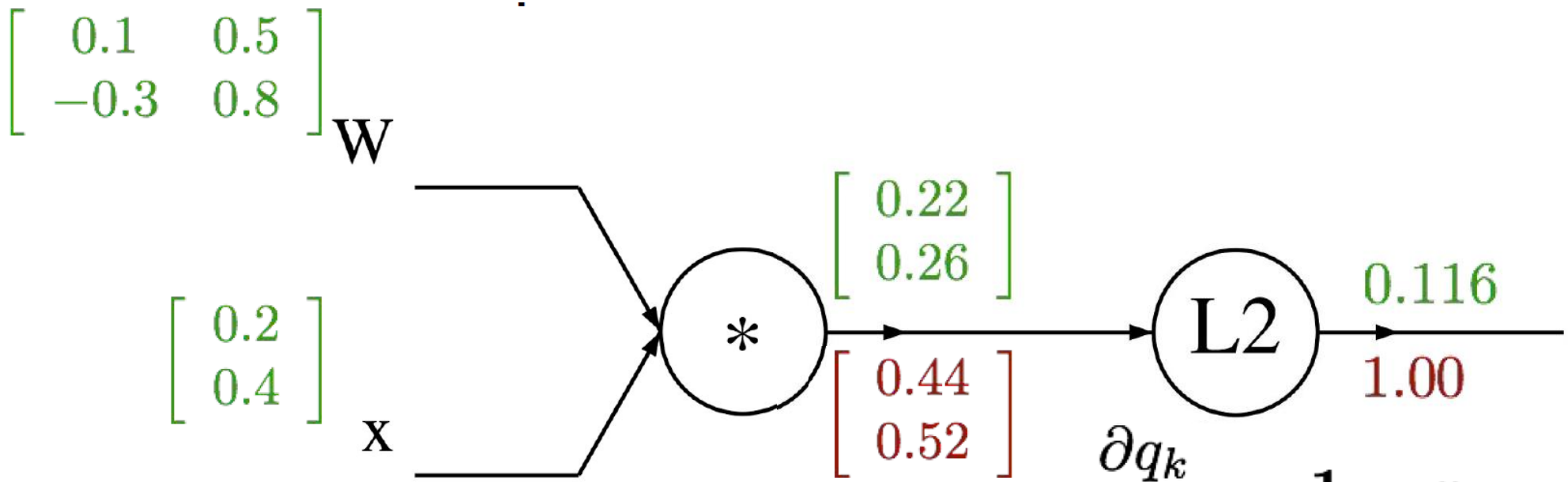
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

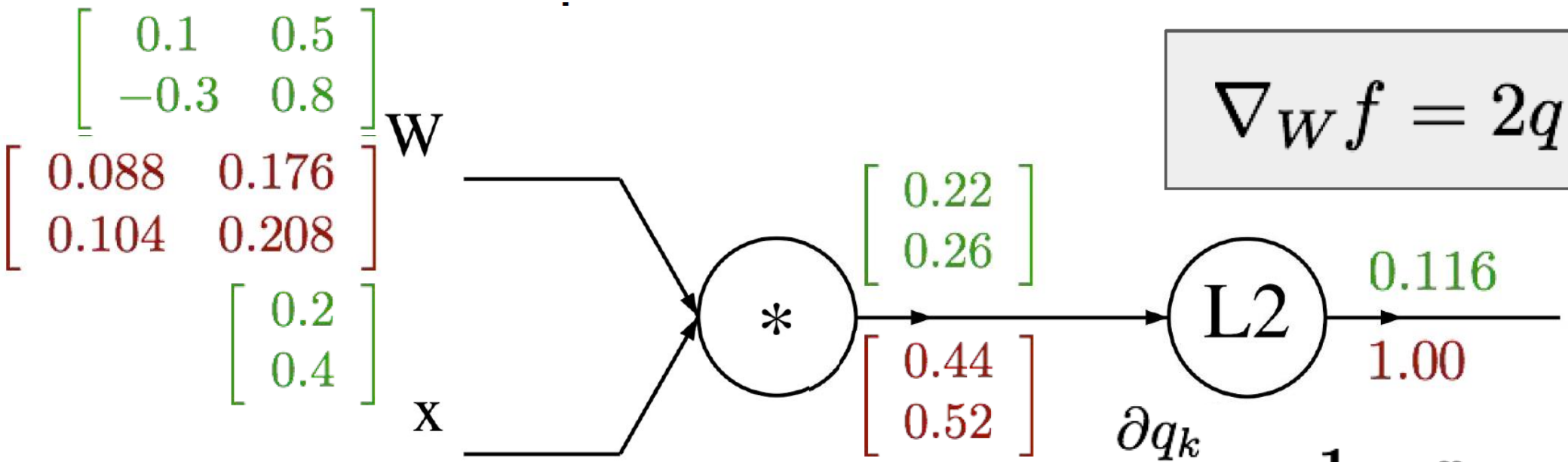
$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$\nabla_W f = 2q \cdot x^T$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

Always check: The gradient with respect to a variable should have the same shape as the Variable

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

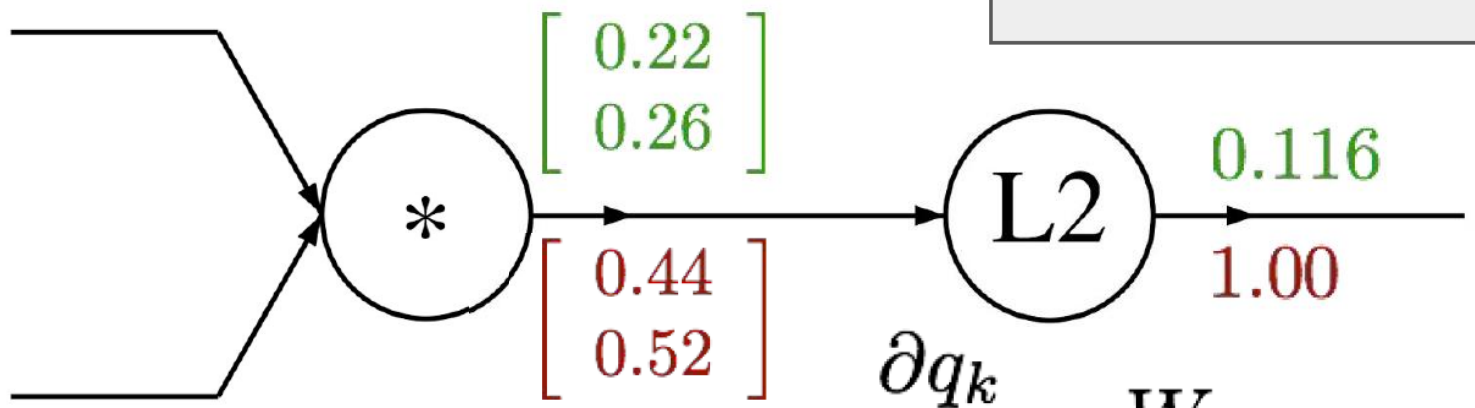
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$ W

$$\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix}$$

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$ x

$$\begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$



$$\begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

$$\nabla_x f = 2W^T \cdot q$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

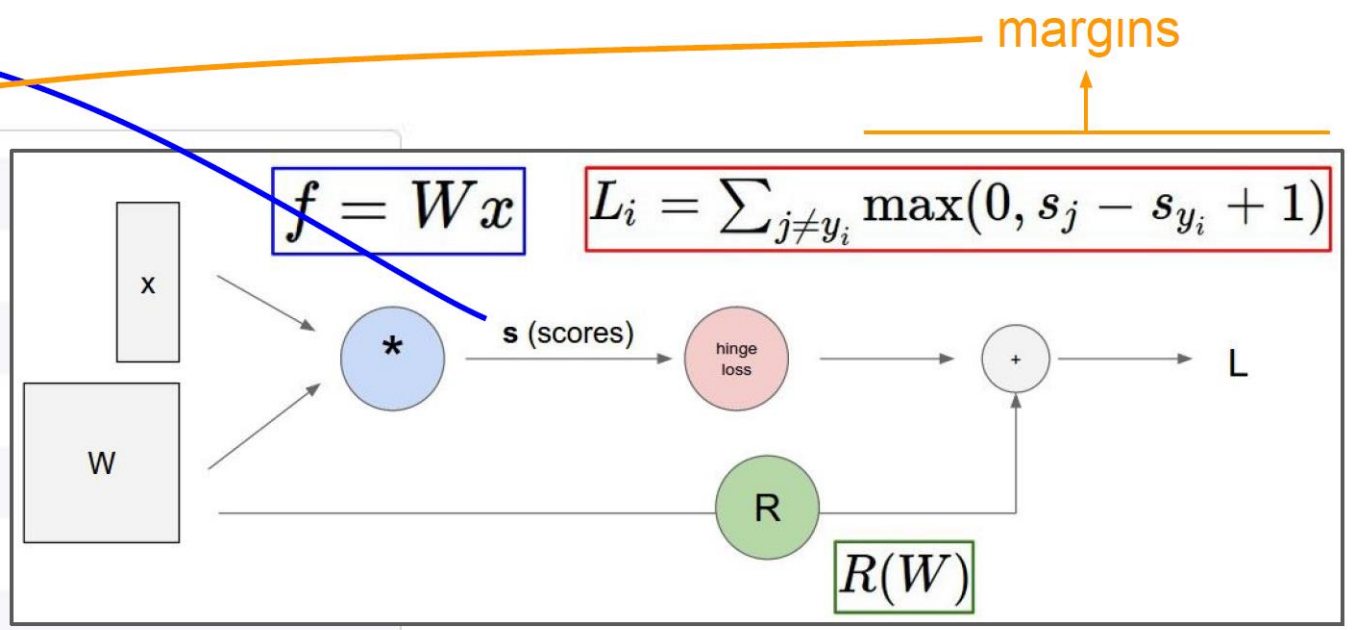
$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i}$$

$$= \sum_k 2q_k W_{k,i}$$

SVM example

E.g. for the SVM:

```
# receive W (weights), X (data)
# forward pass (we have 8 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```



Summary

- Neural nets may be very large: impractical to write 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

Converting error derivatives into a learning procedure

- The backpropagation algorithm is an efficient way of computing the error derivative dE/dw for every weight on a single training case.

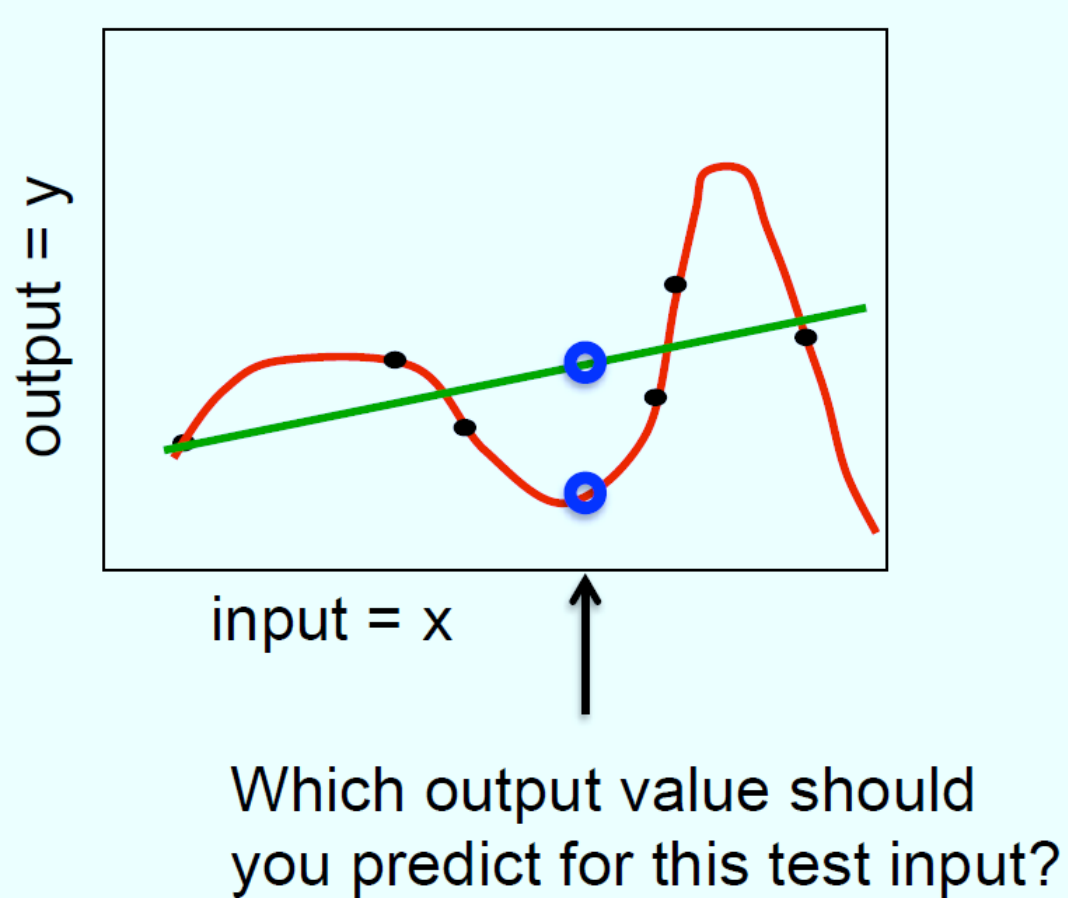
- To get a fully specified learning procedure, we still need to make a lot of other decisions about how to use these error derivatives:
  - Optimization issues: How do we use the error derivatives on individual cases to discover a good set of weights?
  - Generalization issues: How do we ensure that the learned weights work well for cases we did not see during training?

Optimization issues in using the weight derivatives

- How to initialize weights
- How often to update the weights
  - Batch size
- How much to update
  - Use a fixed learning rate?
  - Adapt the global learning rate?
  - Adapt the learning rate on each connection separately?
  - Don’t use steepest descent?
- …

Overfitting: The downside of using powerful models

- A model is convincing when it fits a lot of data surprisingly well.
  - It is not surprising that a complicated model can fit a small amount of data well.



Ways to reduce overfitting

- A large number of different methods have been developed.
  - Weight-decay
  - Weight-sharing
  - Early stopping
  - Model averaging
  - Dropout
  - Generative pre-training

Resources

- Deep Learning Book, Chapter 6.
- Please see the following note:
  - http://cs231n.github.io/optimization-2/