Photogrammetry & Robotics Lab

Intro to Neural Networks
Part 2: Learning

Cyrill Stachniss

The slides have been created by Cyrill Stachniss.

5 Minute Preparation for Today

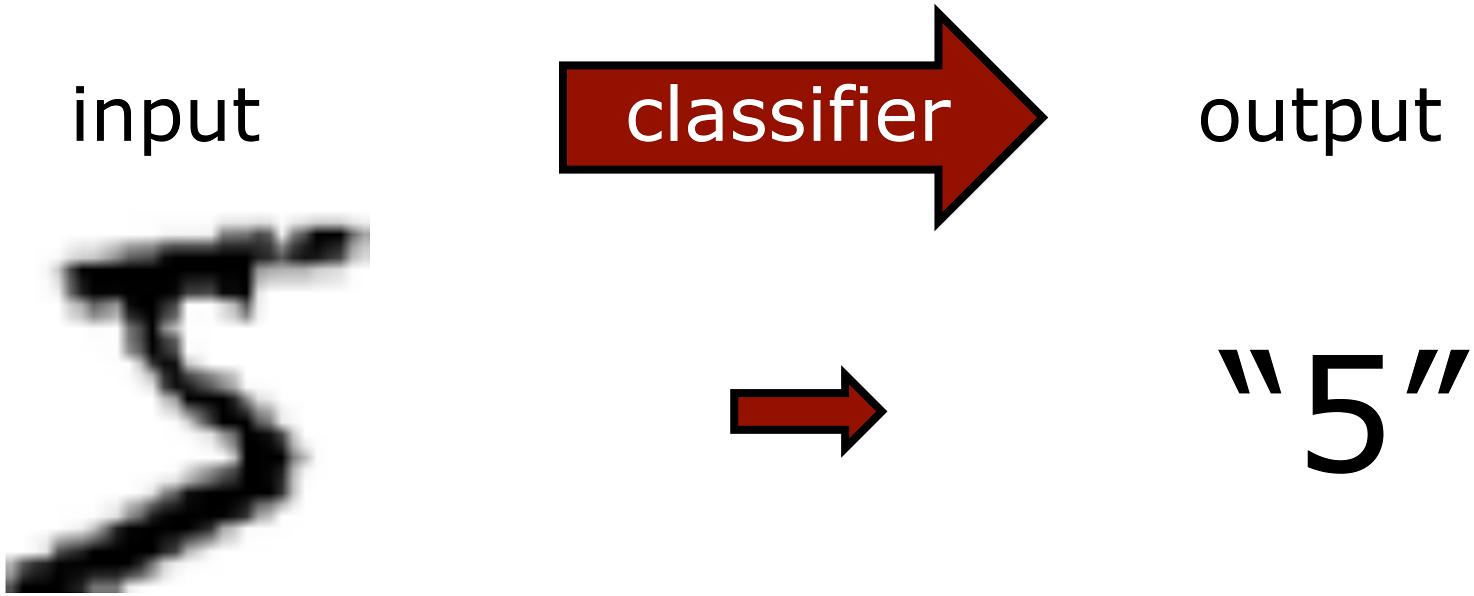
https://www.ipb.uni-bonn.de/5min/

5 Minute Preparation for Today

https://www.ipb.uni-bonn.de/5min/

In Part 1, We Discussed

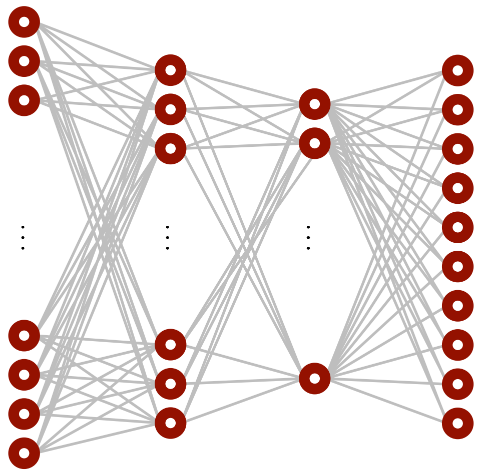
- What are neurons and neural networks
- Activations, weights, biases
- Multi-layer perceptron (MLP)
- MLP for simple image classification



Part 2
Learning the Parameters

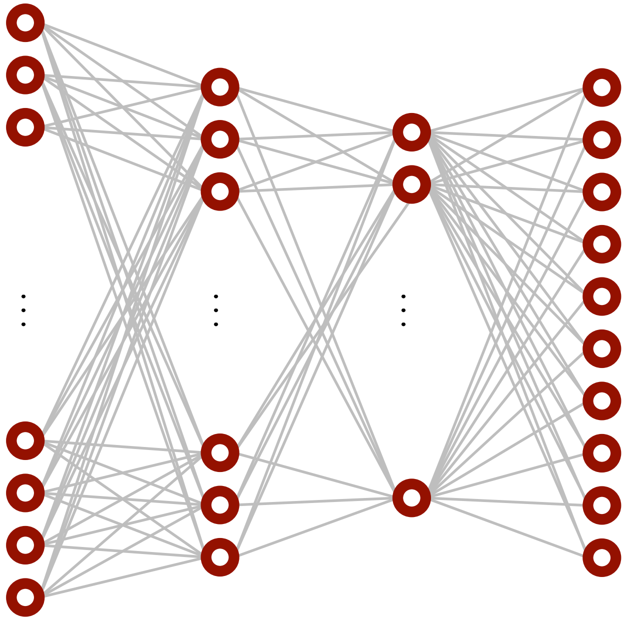
How to Make the Network Compute What We Want?

- Neural network is a recipe for performing a set of computations
- Structure and parameters are the design choices
- **How to set them?**



Network Parameters

Given a network structure, weights and biases tell the network what to do



input

$$\sigma\left(W^{(1)}\boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)}\right) = \boldsymbol{a}^{(1)}$$

$$\sigma\left(W^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}\right) = \boldsymbol{a}^{(2)}$$

$$\sigma\left(W^{(k)}\boldsymbol{a}^{(k-1)} + \boldsymbol{b}^{(k)}\right) = \boldsymbol{a}^{(k)}$$

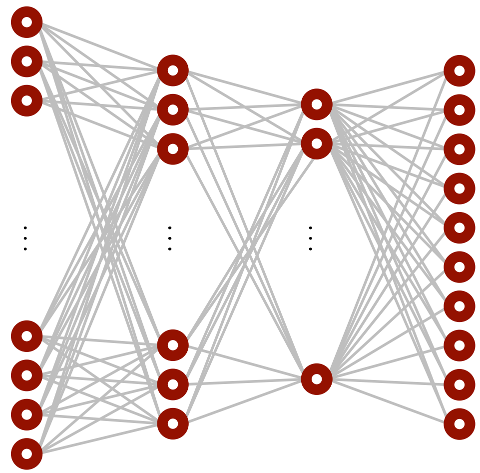
output

weights biases

params = weights & biases $\boxed{W^{(1)}\ \boldsymbol{b}^{(1)}\ \dots\ W^{(k)}\ \boldsymbol{b}^{(k)}}$

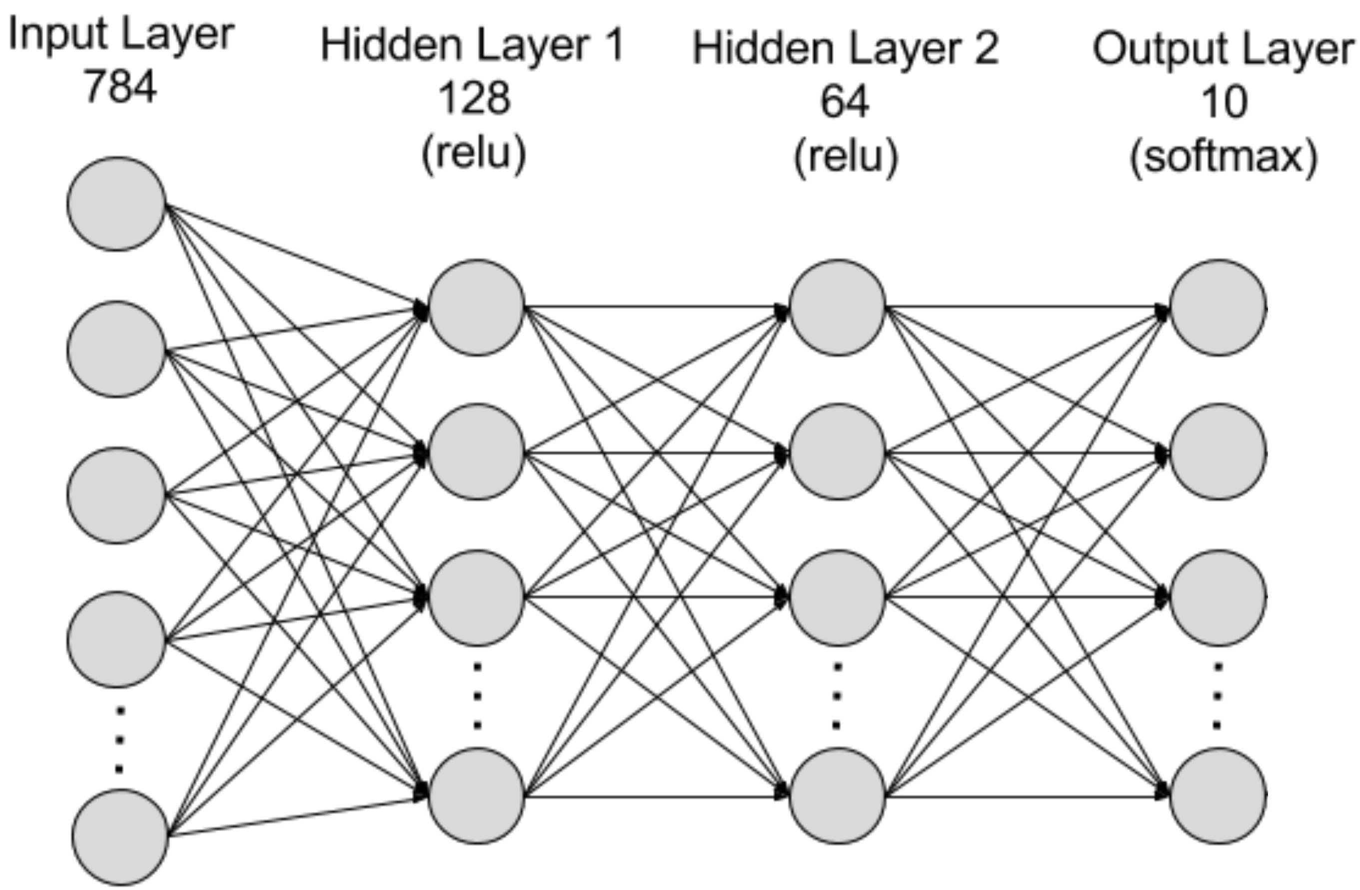
What Means “Learning”?

- NN are functions
- The weights and biases determine what this function computes



- Learning = determining parameters so that the network does what we want
- Parameters are estimated by providing labeled examples (training data)

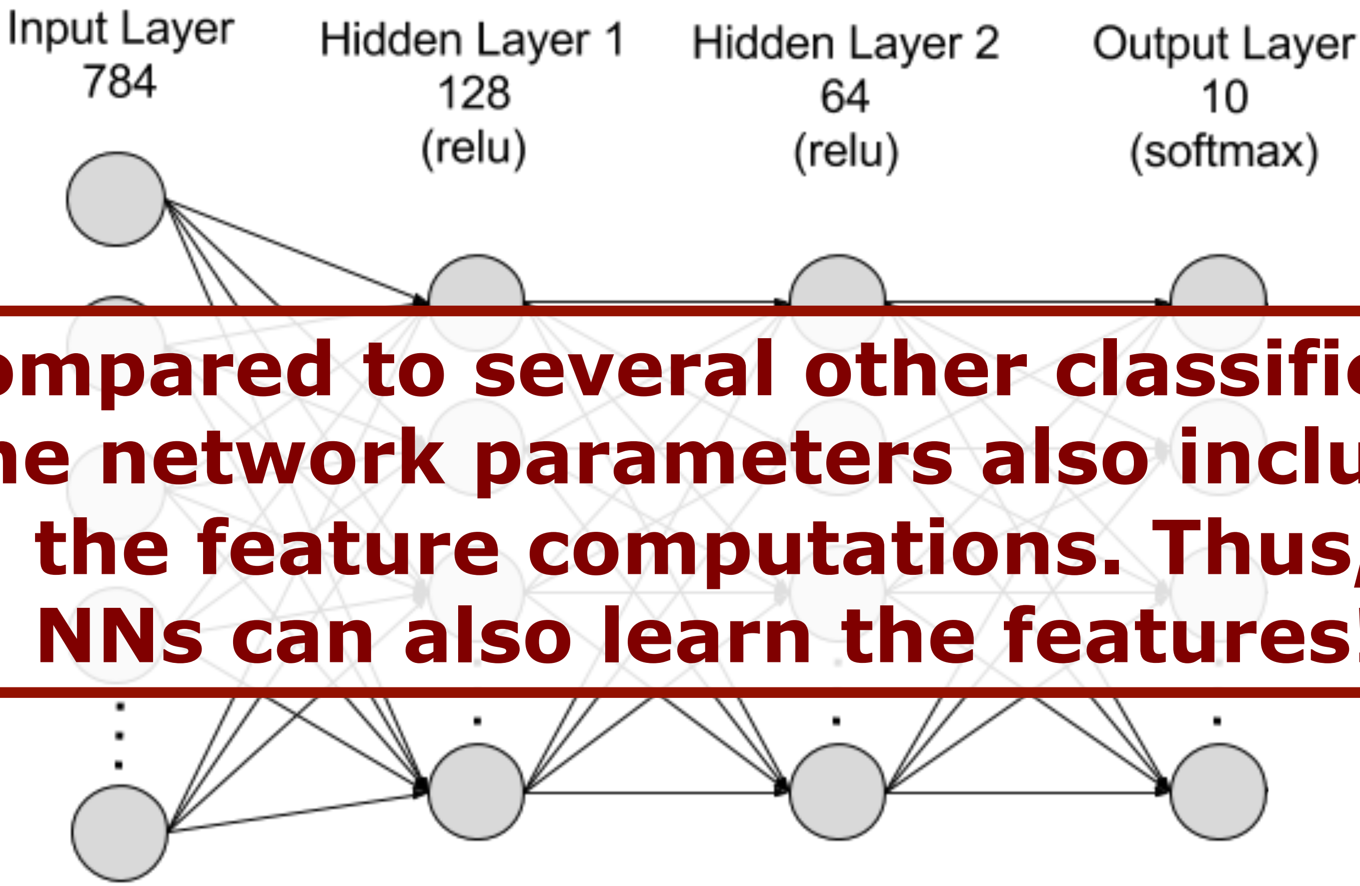
Our Handwritten Digit Network



raw pixels | **simple patterns** | **combined patterns** | **patterns to digits**

[Image courtesy: Nielsen]

Parameters Encode Features



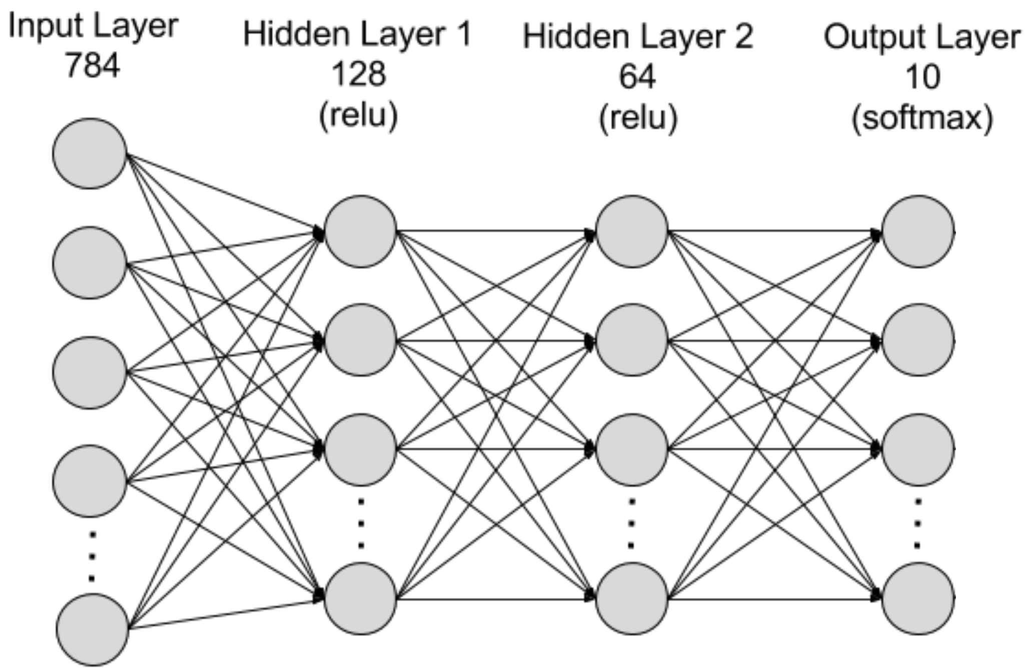
Compared to several other classifiers, the network parameters also include the feature computations. Thus, NNs can also learn the features!

raw pixels	**simple patterns**	**combined patterns**	**patterns to digits**

[Image courtesy: Nielsen]

Many Parameters

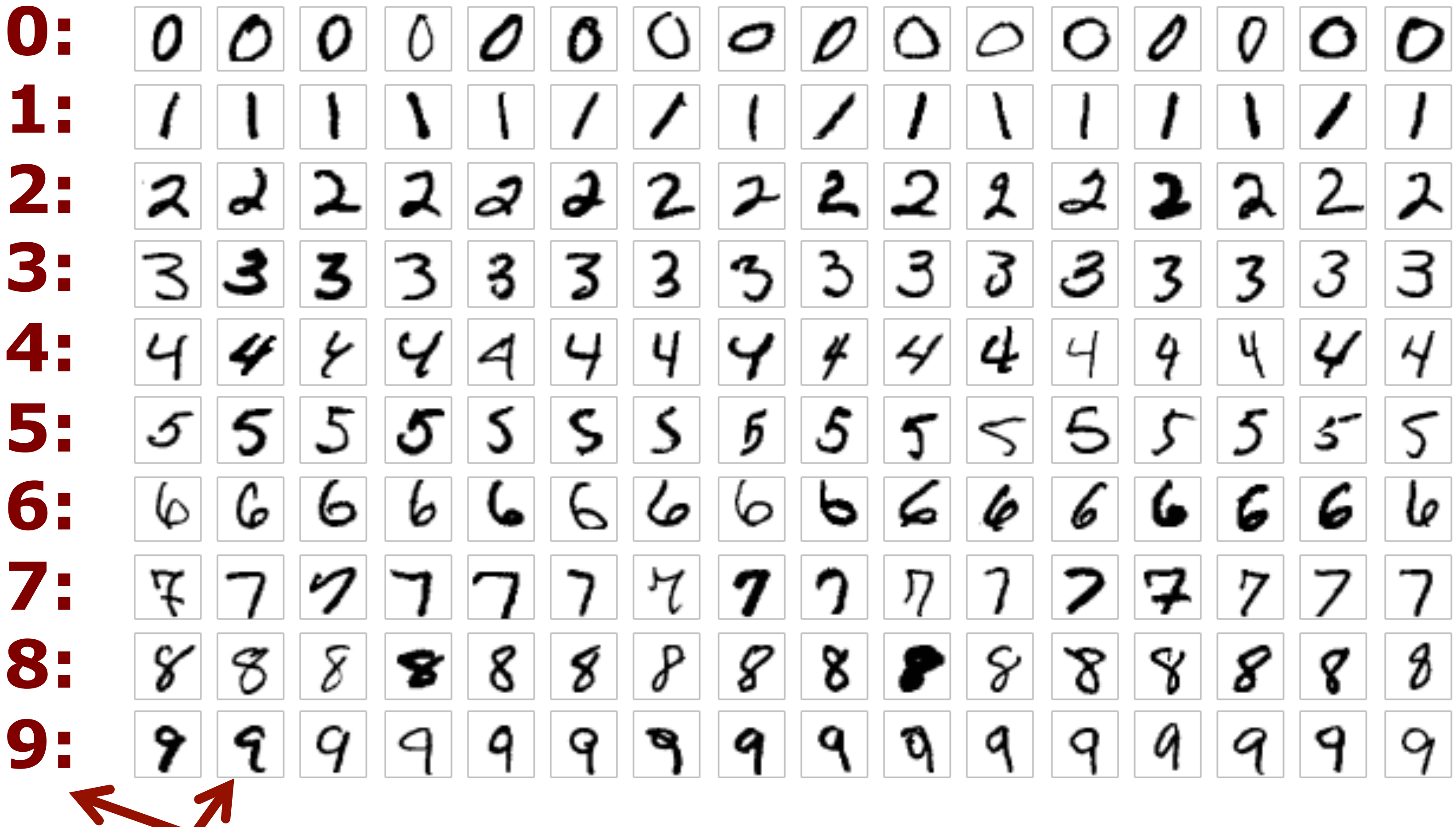
Such networks have **many** parameters!



$$W^{(1)} \quad \boldsymbol{b}^{(1)} \quad W^{(2)} \quad \boldsymbol{b}^{(2)} \quad W^{(3)} \quad \boldsymbol{b}^{(3)}$$

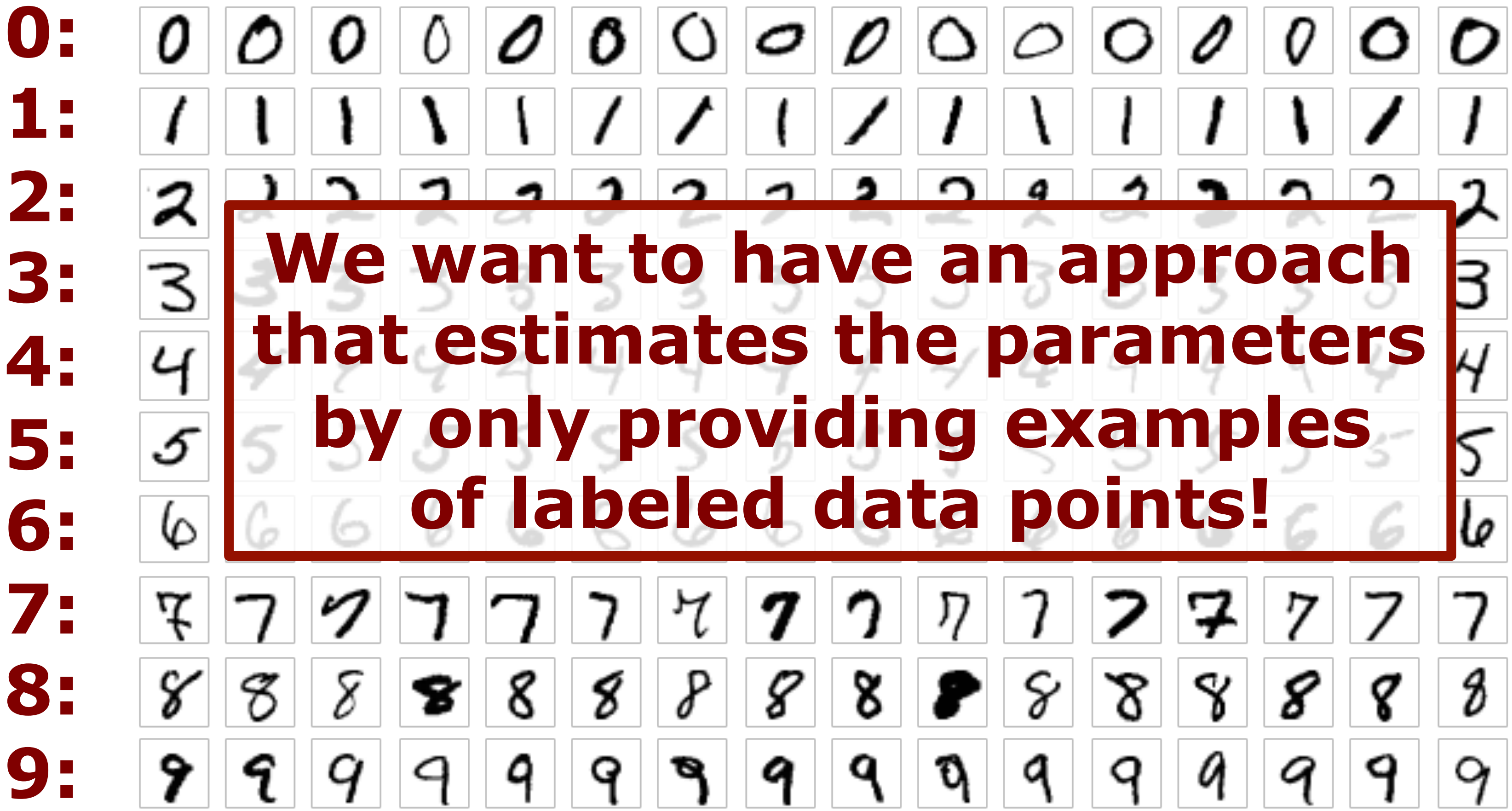
784x128 128 128x64 64 64x10 10 **= 109.386 parameters**

Training Through Labeled Data

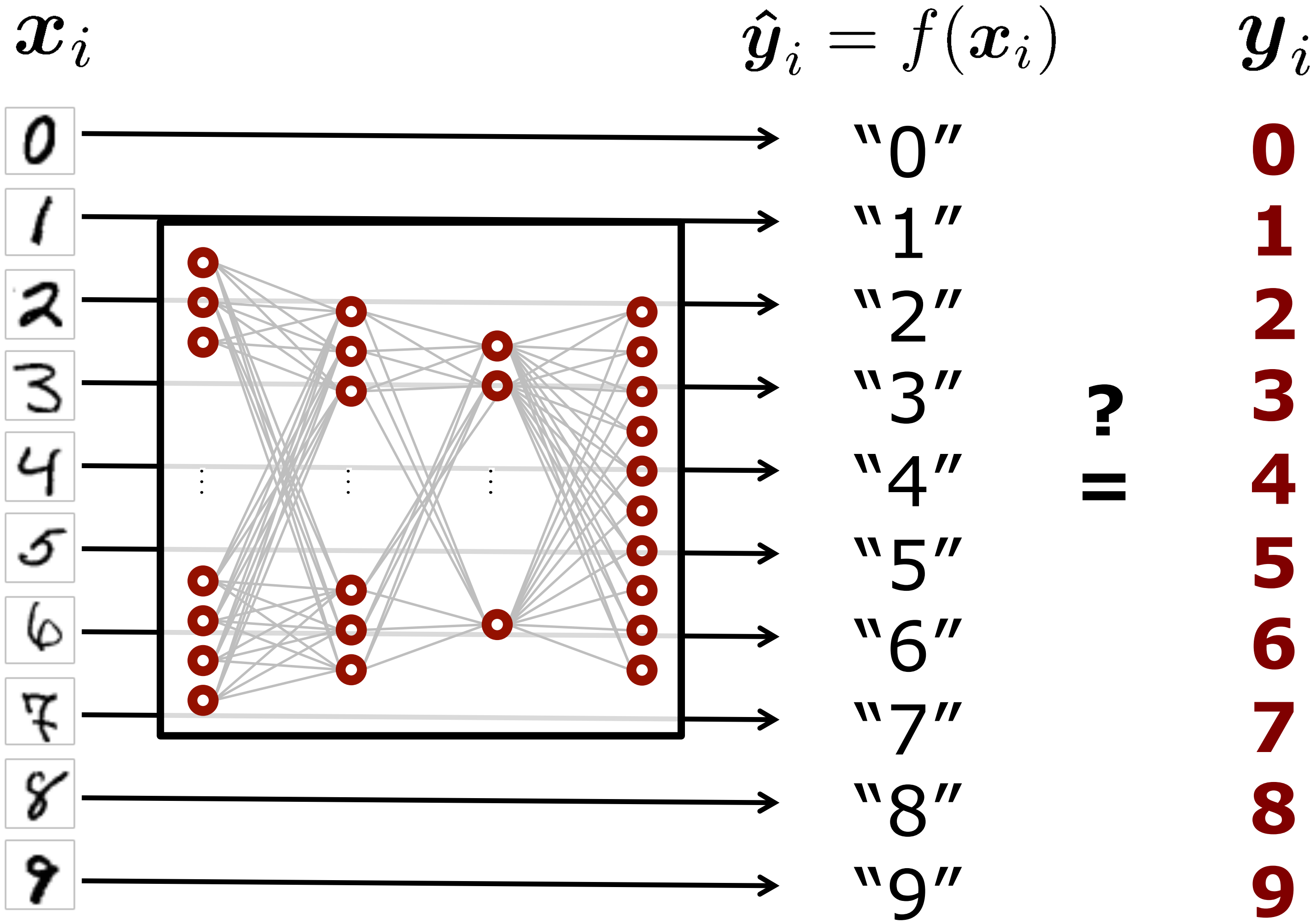


$$\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{I}$$

Training Through Labeled Data



Exploiting Training Examples



Loss Function

- We define a loss (= cost) function over all weights and biases of the network

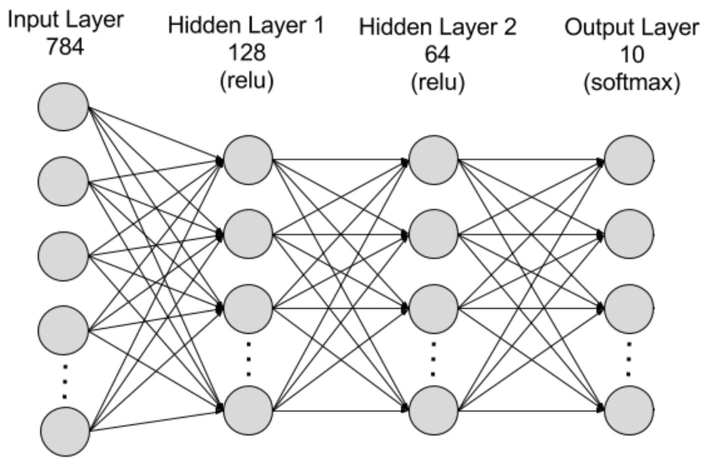
$$L\left(W, \boldsymbol{b}\right) \mapsto \mathbb{R}$$

- Computed using training data
- Input to L are the network parameters $\boldsymbol{\theta} = W, \boldsymbol{b}$
- Output is the error of the network with these parameters on the training data

Loss Function $L_i(\boldsymbol{\theta}) \mapsto \mathbb{R}$

$\boldsymbol{x}_i$

data point



network with parameters θ

$\hat{\boldsymbol{y}}_i = f(\boldsymbol{\theta}, \boldsymbol{x}_i)$

$$\begin{bmatrix} 0.2 \\ 0 \\ 0.3 \\ 0.2 \\ 0.2 \\ 0.9 \\ 0 \\ \vdots \end{bmatrix}$$

network output

$\boldsymbol{y}_i$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

label

Compare output layer to the true label

$$L_i(\boldsymbol{\theta}) = ||\text{output}_i - \text{label}_i||^2$$

Loss Over All Examples

We need to evaluate the performance of the network over all examples

loss = all examples, avg. squared(NN(input) – label)

$$L(\boldsymbol{\theta}) = \frac{1}{I}\sum_i L_i(\boldsymbol{\theta}) = \frac{1}{I}\sum_i ||f(\boldsymbol{\theta}, \boldsymbol{x}_i) - \boldsymbol{y}_i||^2$$

{(, 5), (, 2), (, 2), (, 9), (, 8), ...

$$\{(\boldsymbol{x}_0, \boldsymbol{y}_0), (\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots$$

The Parameters We Want

- Parameter $\boldsymbol{\theta}^*$ that minimize the sum of avg. squared losses over all examples

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_i ||f(\boldsymbol{\theta}, \boldsymbol{x}_i) - \boldsymbol{y}_i||^2$$

- The squared loss is only one possible loss, several other options available
- **Goal:** Find the parameter vector $\boldsymbol{\theta}^*$ for the labeled training set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{I}$ given the loss L

Let’s Start…

- Initialize parameters randomly
- See how well it performs (bad!)
- How to improve the parameters so that the loss decreases?

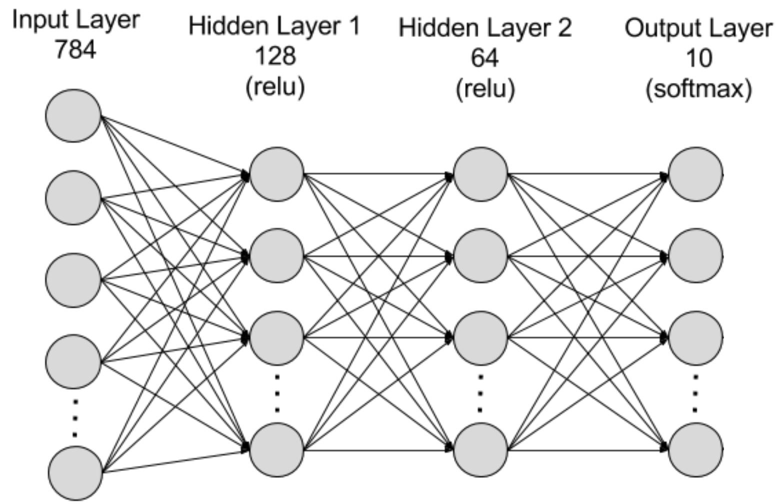
$$L(\boldsymbol{\theta}) \mapsto \mathbb{R}$$

high dimensional

1 dim

(109.386 dim)

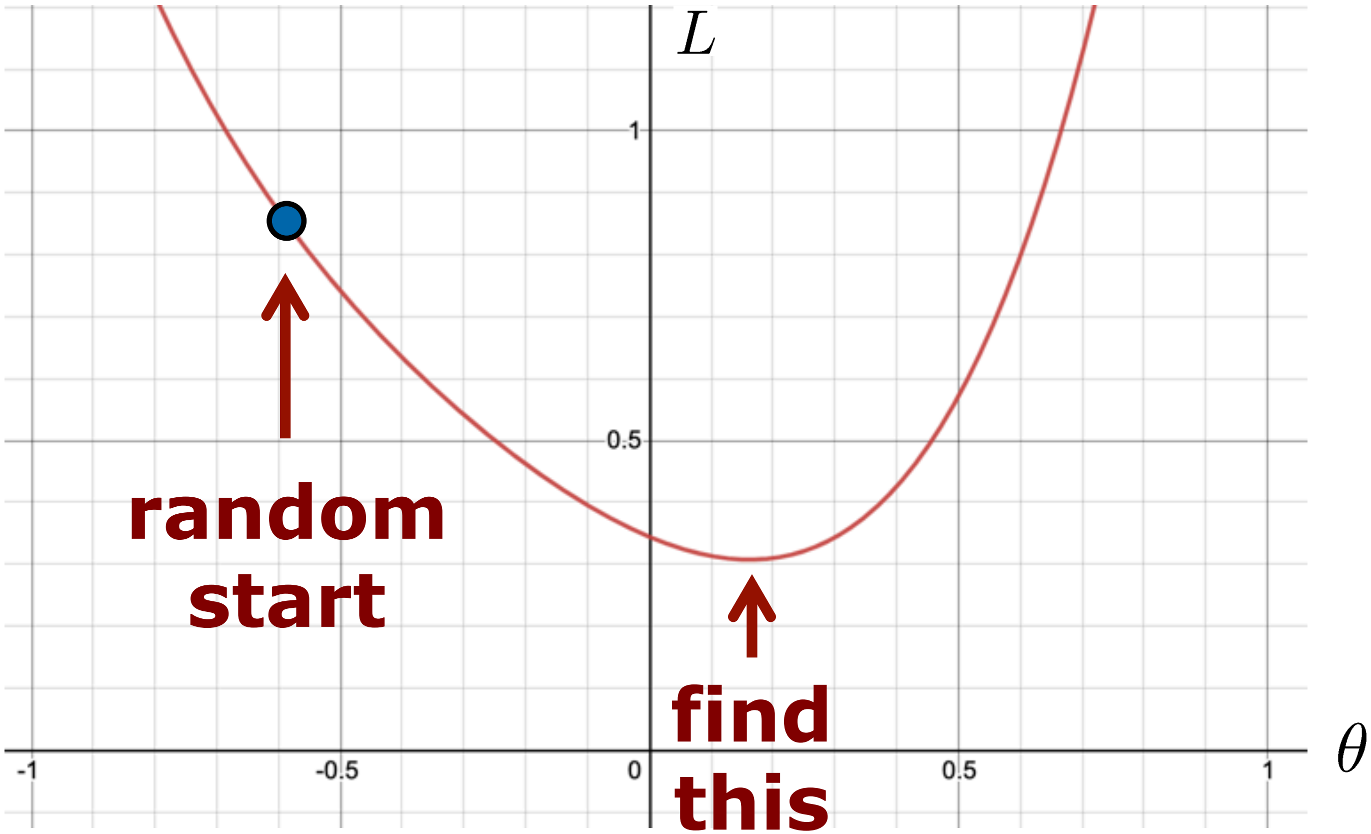
That’s complex!



Loss Minimization using Gradient Descent

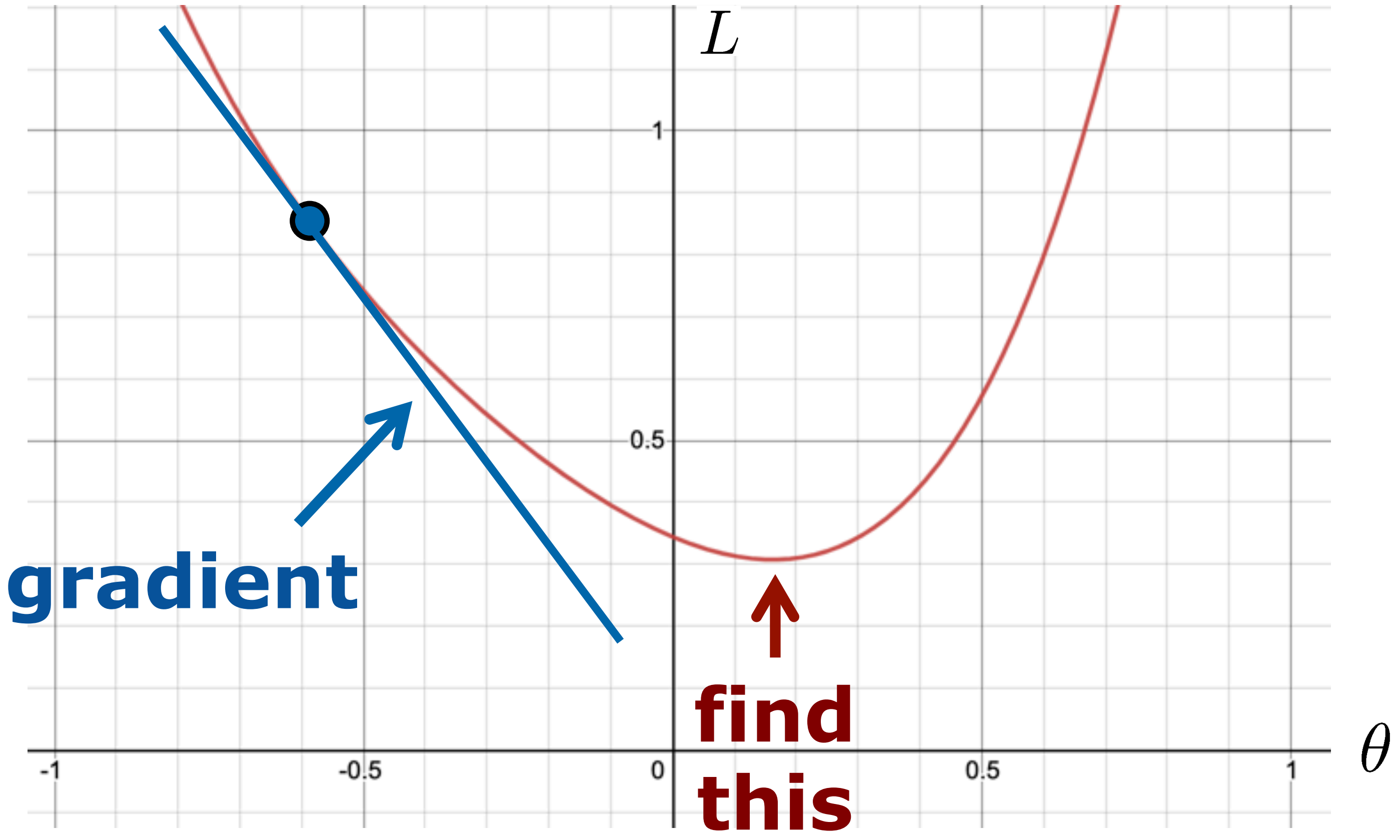
- Our problem looks like a non-linear least squares problem
- We have **a lot** of parameters, which makes using GN computationally tricky
- Gradient descent is a better way to perform the minimization

Gradient Descent in 1D



How to move?

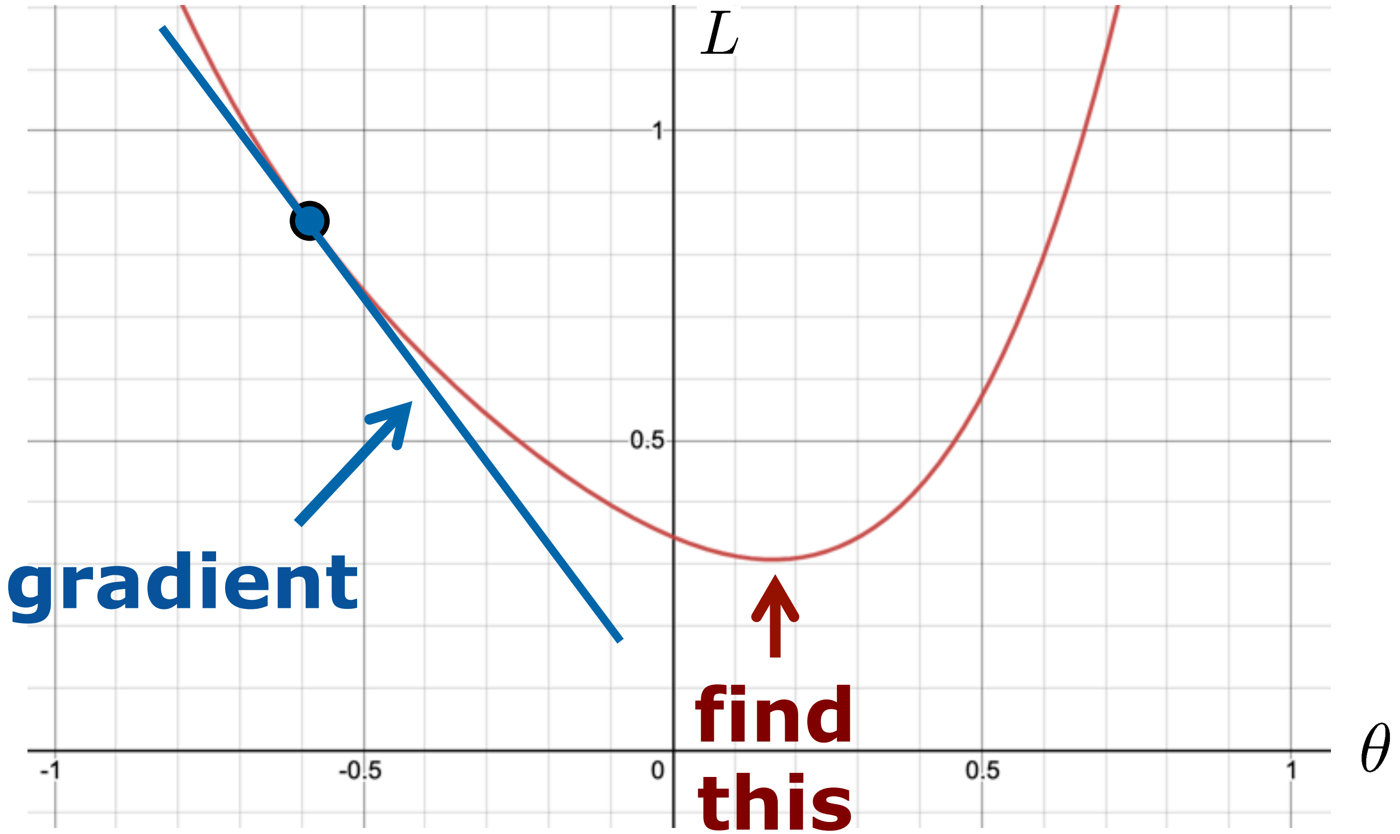
Gradient Descent in 1D



How to move?

Exploit the gradient

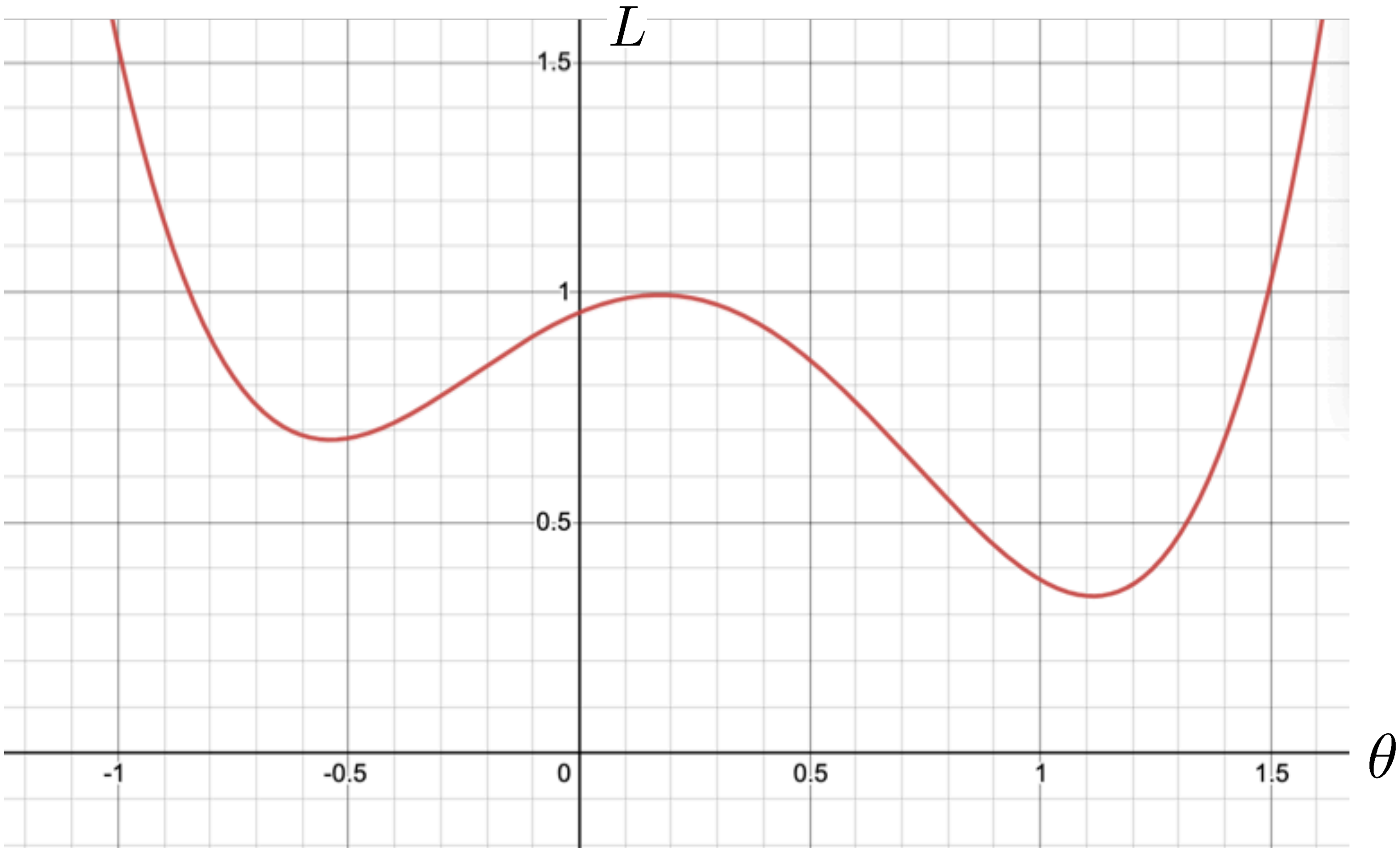
Gradient Descent in 1D



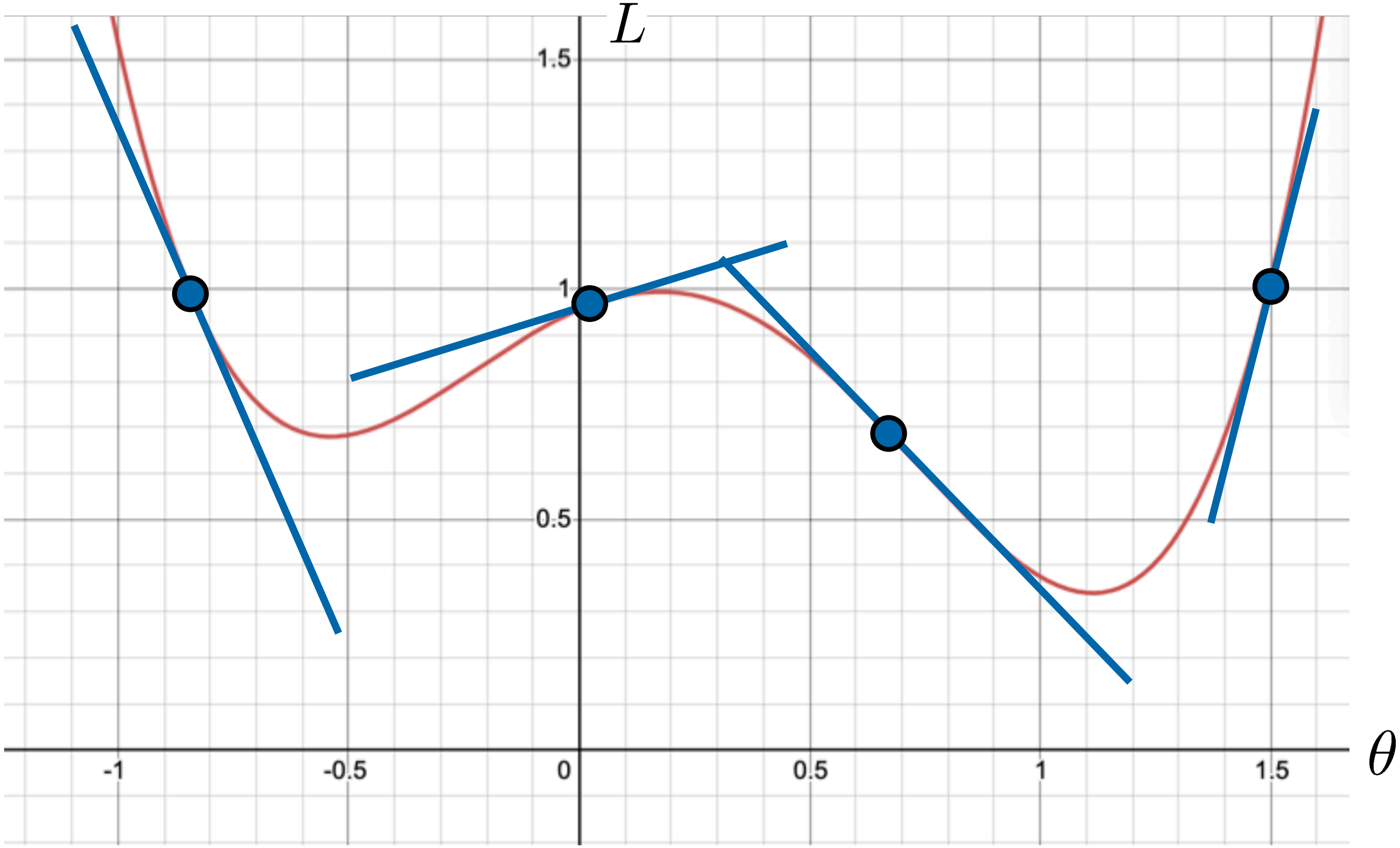
Strategy:

$$\frac{\partial L}{\partial \theta} > 0 \quad : \quad \leftarrow \quad \text{(move left)}$$
$$\frac{\partial L}{\partial \theta} < 0 \quad : \quad \rightarrow \quad \text{(move right)}$$

Gradient Descent in 1D

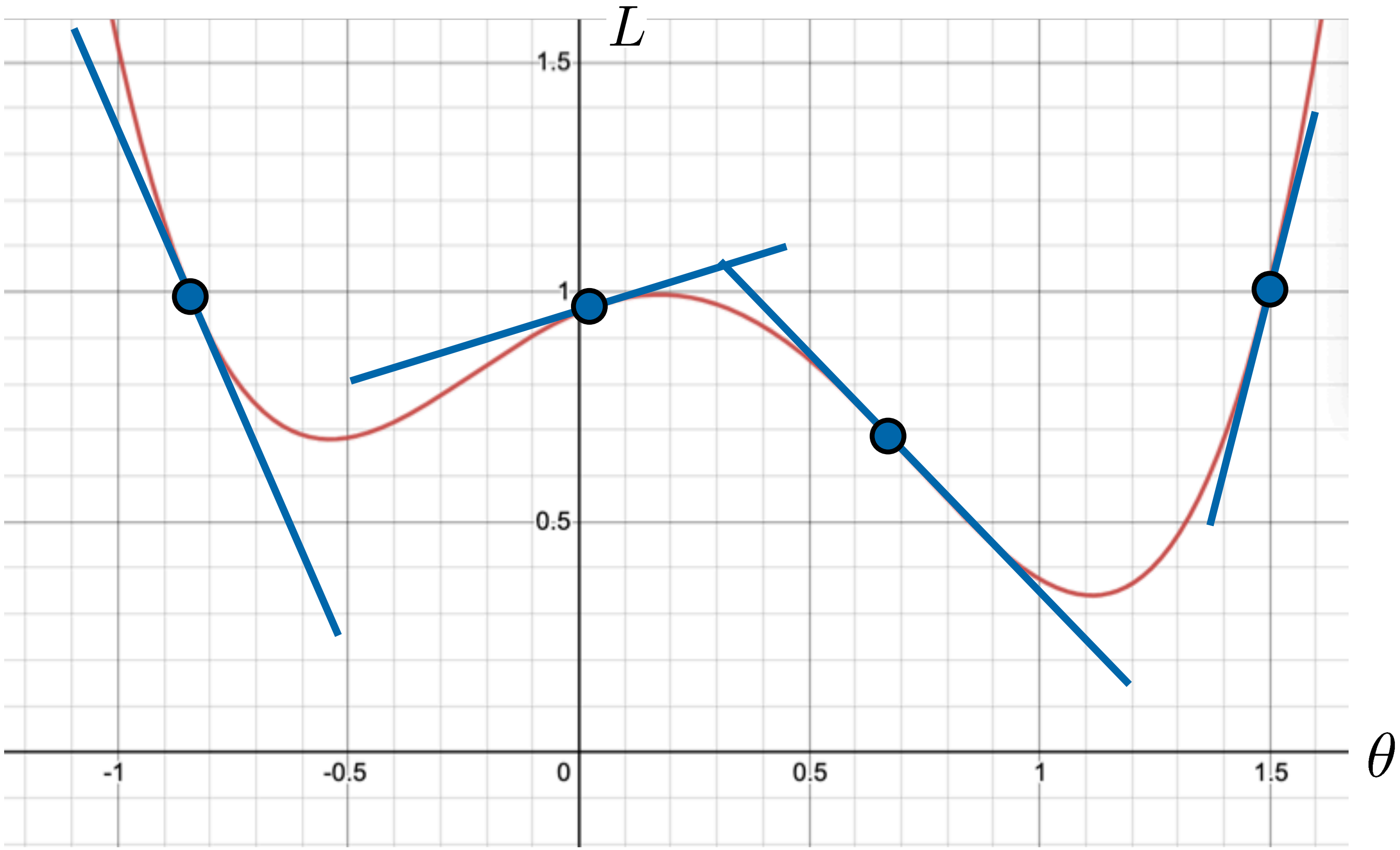


Gradient Descent in 1D



$$\nabla L = \frac{\partial L}{\partial \theta}$$ **tells us in which direction to go**

Gradient Descent in 1D



Step by step: $\theta^{(j+1)} = \theta^{(j)} - \lambda \, \nabla L|_{\theta^{(j)}}$

Gradient Descent in 1D

- First derivative of loss: $\nabla L = \frac{\partial L}{\partial \theta}$
- Learning rate (small value): $\lambda = 0.01$

Gradient descent works by

- $\theta^{(0)} = \text{rand}$
- while (!converged)

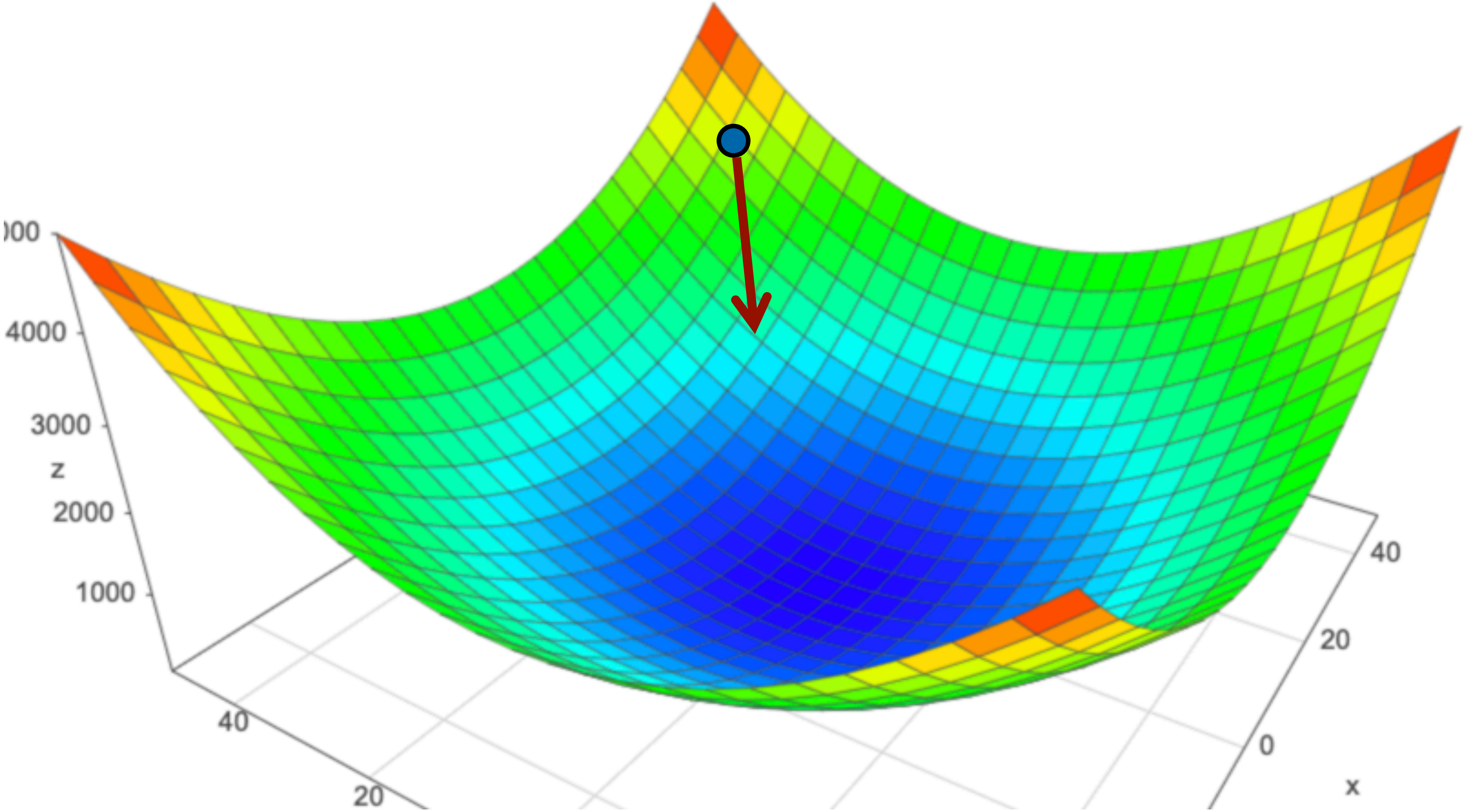
$$\theta^{(j+1)} = \theta^{(j)} - \lambda \, \nabla L|_{\theta^{(j)}}$$

Gradient Descent in 1D

Step by step: $\theta^{(j+1)} = \theta^{(j)} - \lambda \, \nabla L|_{\theta^{(j)}}$

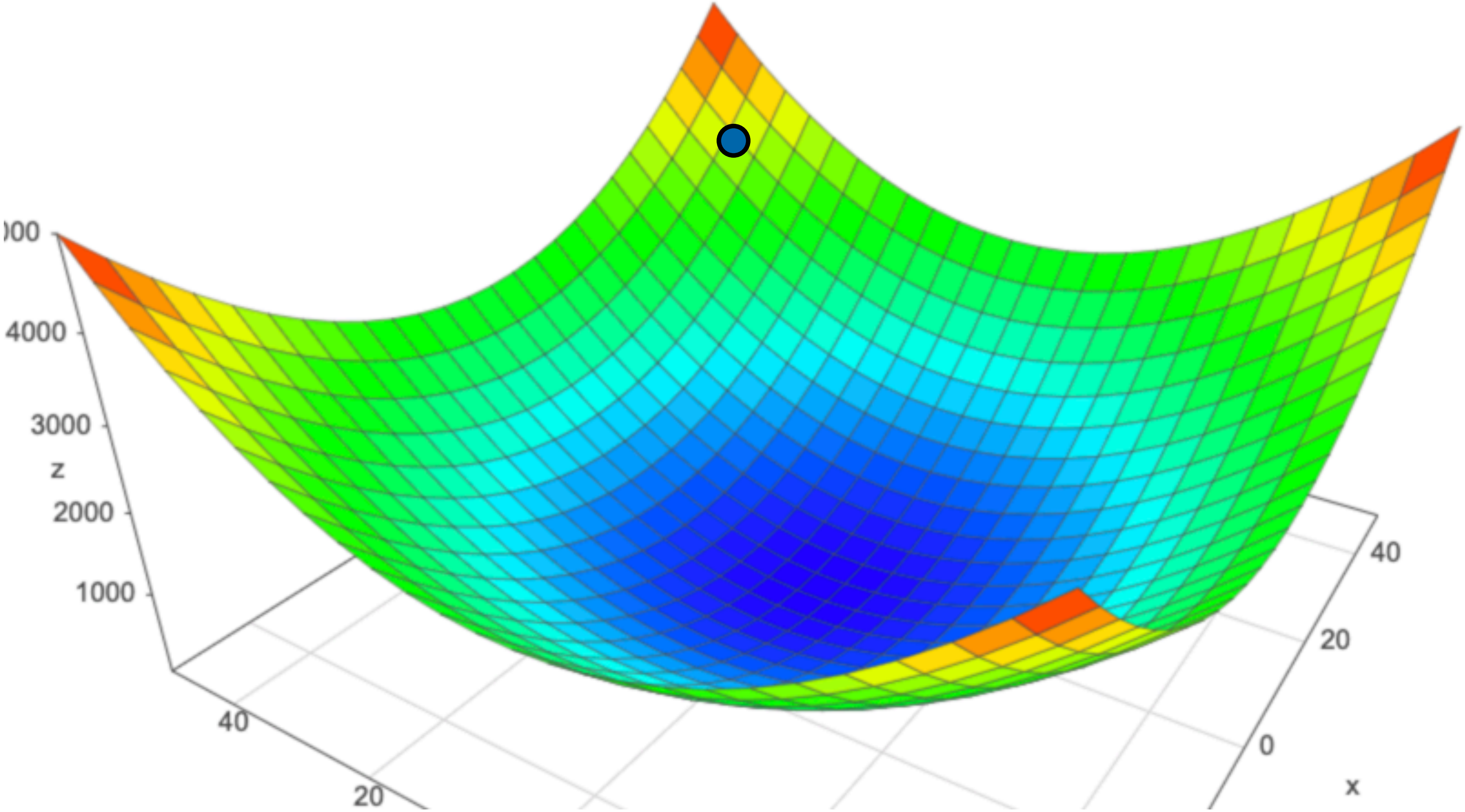
Gradient Descent in 2D

- We can do the same in 2D
- Gradients are direction vectors



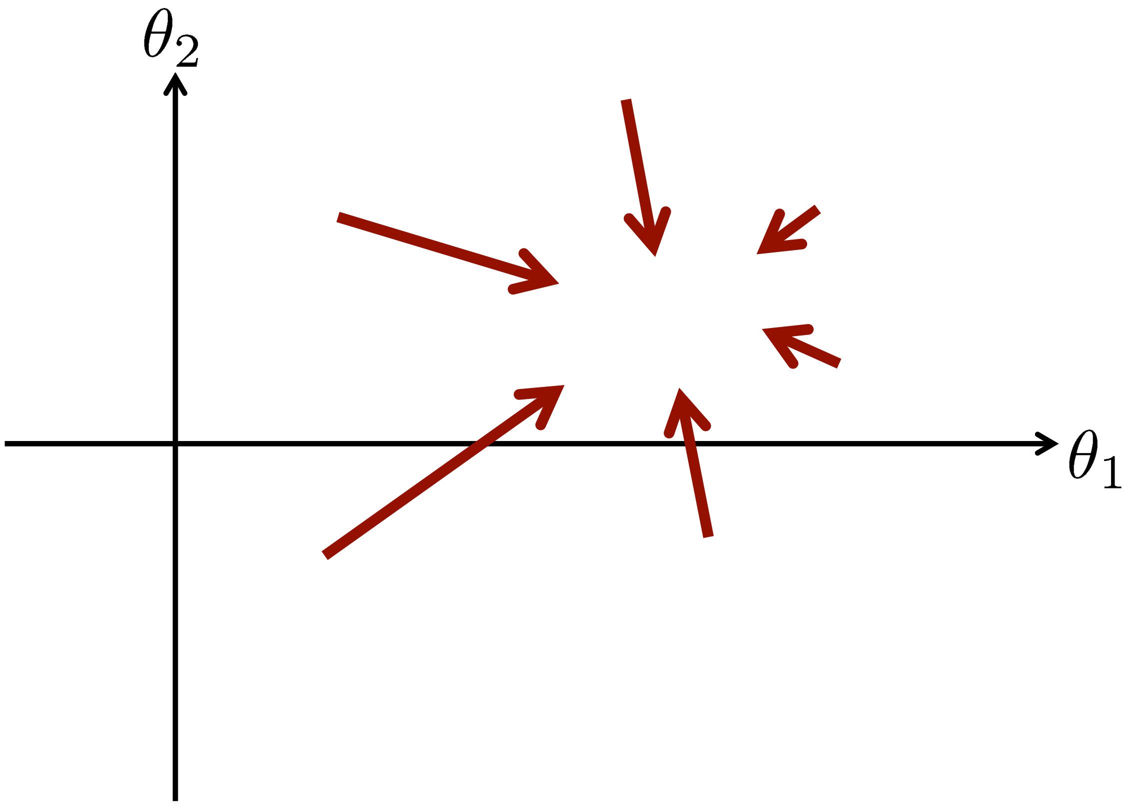
Gradient Descent in 2D

- We can do the same in 2D
- Gradients are direction vectors



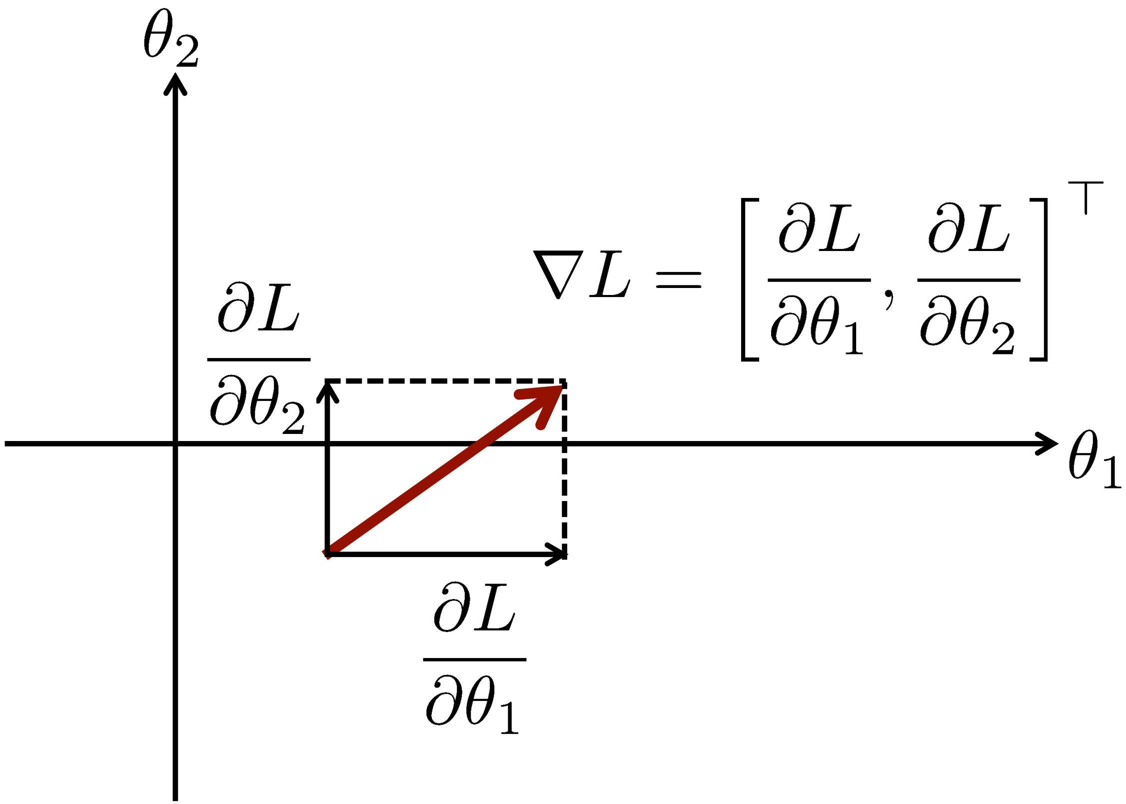
Gradient Descent in 2D

- We can do the same in 2D
- Gradients are direction vectors



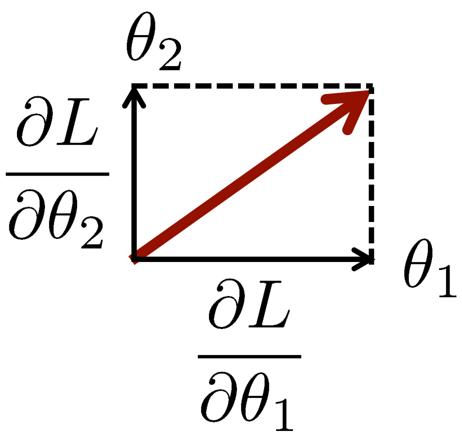
Gradient Descent in 2D

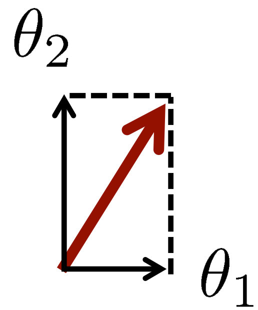
- We can do the same in 2D
- Gradients are direction vectors



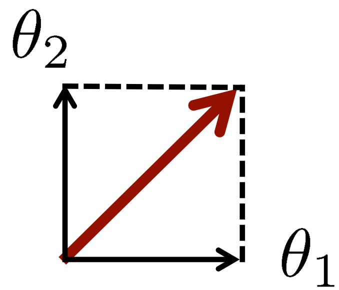
Meaning of the Gradient Vector

- Some dimensions are more important than others to reduce the loss
- Gradient indicates which change leads to the fastest reduction of the loss





changes in θ_2 are more relevant than changes in θ_1 to reduce loss



changes in θ_1 and θ_2 have the same relevance to reduce loss

Gradient Descent in Higher Dimensional Spaces

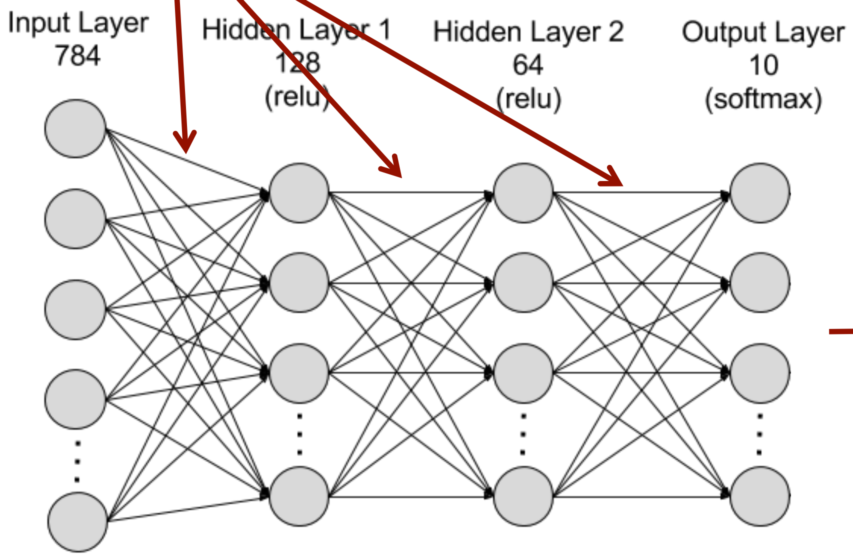
- Same situation as before, but the gradient vector has more dimensions
- Update rule

$$\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} - \lambda \, \nabla L\big|_{\boldsymbol{\theta}^{(j)}}$$

In our classification example, all these vectors have 109.386 dimensions!

Keep in Mind...

$\boldsymbol{\theta}$ parameters are weights and biases



difference between what we get and what we want

$$L_i(\boldsymbol{\theta}) = ||f(\boldsymbol{\theta}, \boldsymbol{x}_i) - \boldsymbol{y}_i||^2$$

total loss is the averaged loss of all examples

$$L(\boldsymbol{\theta}) = \frac{1}{I}\sum_i L_i(\boldsymbol{\theta})$$

We need to adjust the parameters to minimize the total loss!

Gradient Over All Examples

- Loss function sums over all examples

$$L(\boldsymbol{\theta}) = \frac{1}{I}\sum_i L_i(\boldsymbol{\theta})$$

- This means for the gradient

$$\nabla L = \frac{1}{I}\sum_i \nabla L_i$$

We need to sum over all training examples and compute all gradients whenever we perform a single GD step!

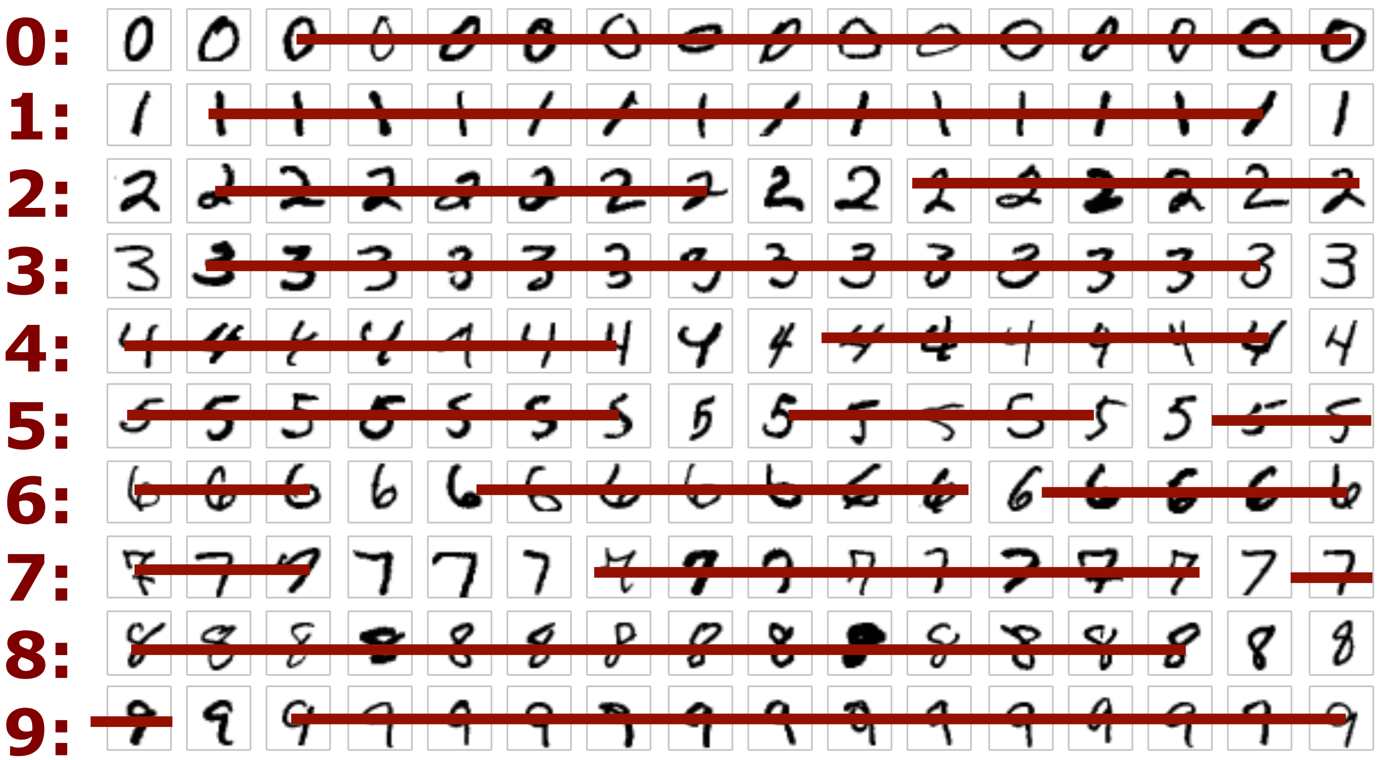
Two Challenges

1. How to optimize the process if we have a **lot of training examples**?
2. How to compute the gradients for **complex and nested functions**?

We need to perform these operations often, so we need to be able to execute them efficiently!

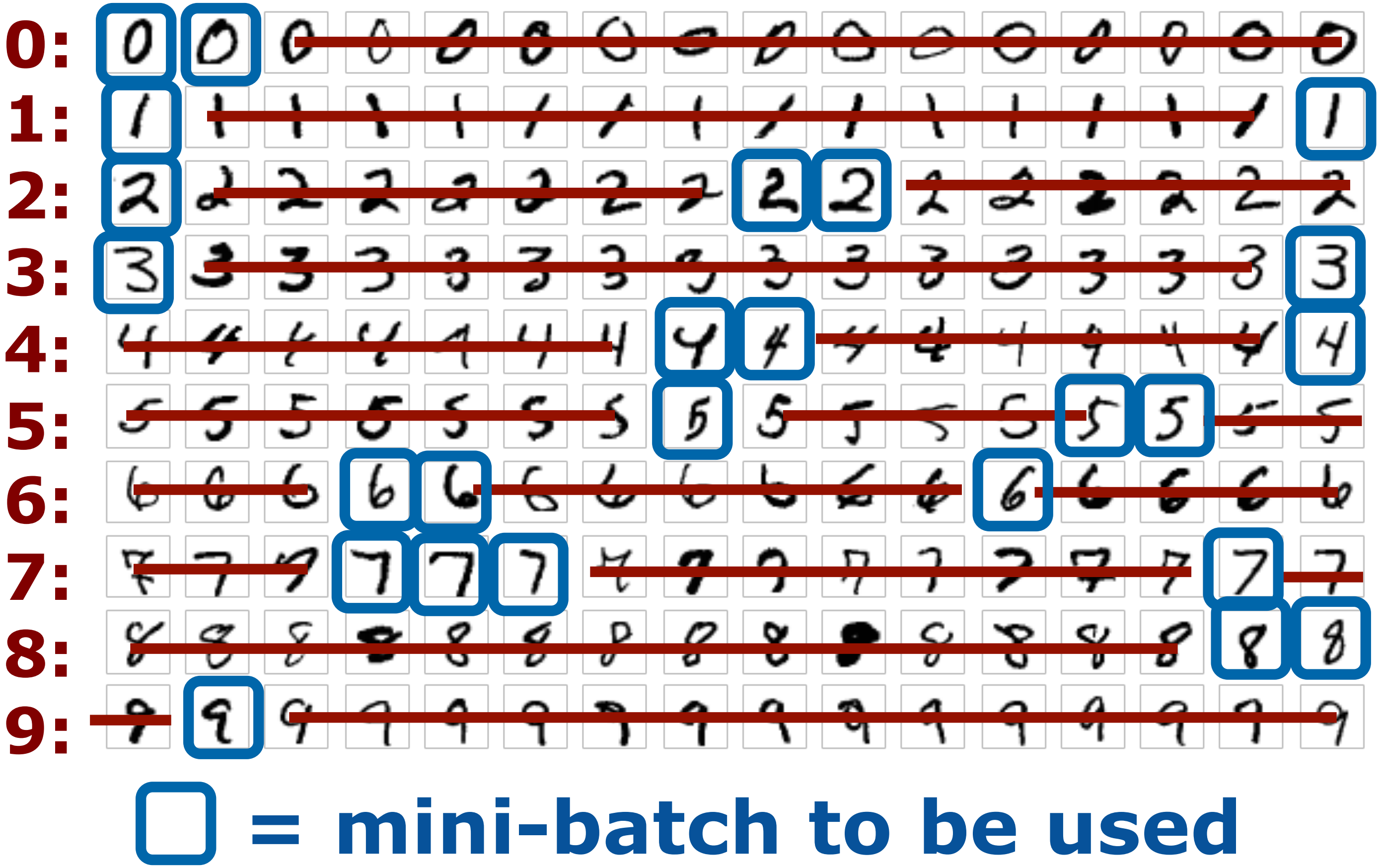
1: Handling Large Training Sets

1st trick: Compute a gradient only on a small, sampled subset of examples



1: Handling Large Training Sets

1st trick: Compute a gradient only on a small, sampled subset of examples



1: Stochastic Gradient Descent

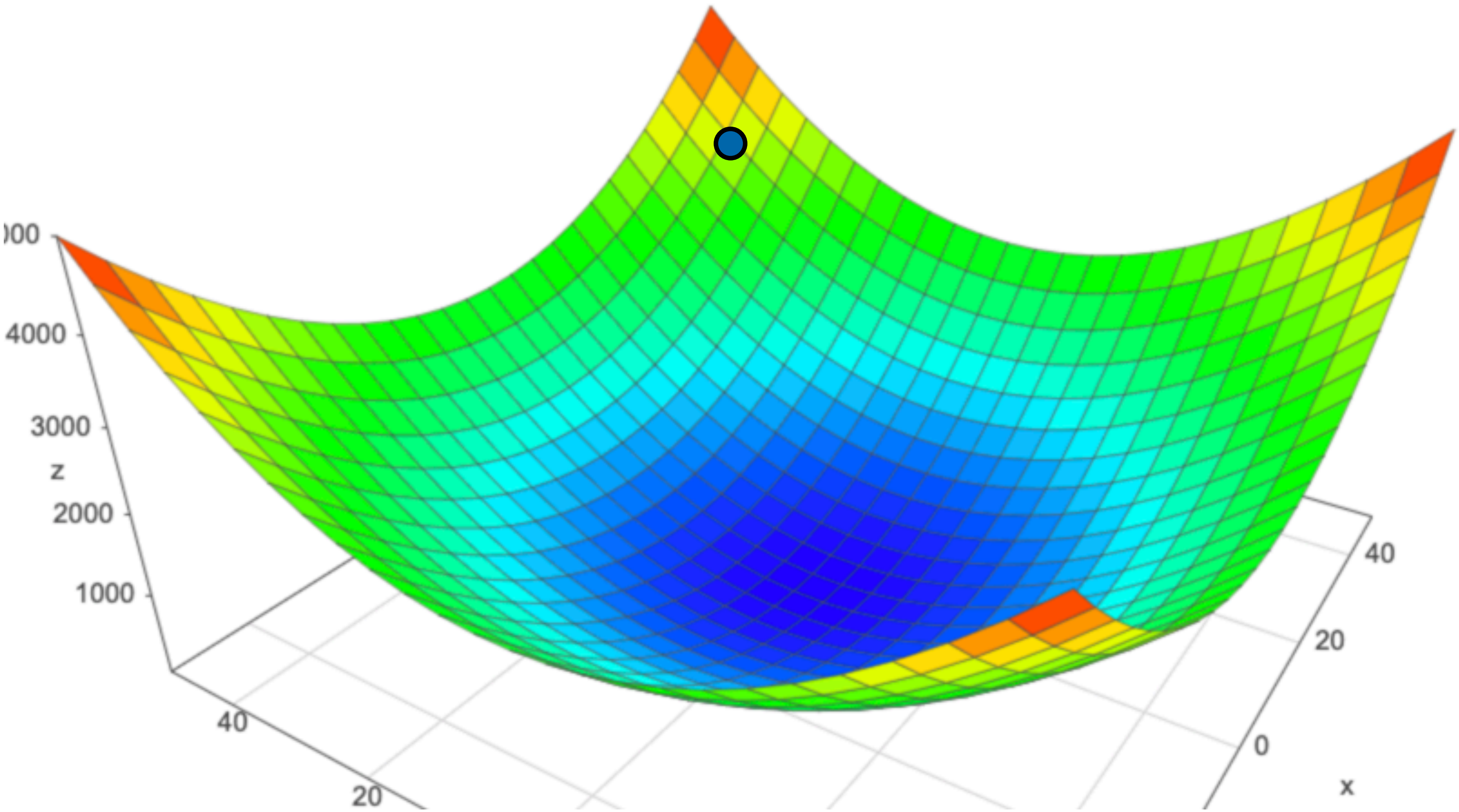
- 1st trick: Compute a gradient only on a small, sampled subset of examples
- We **sample a mini-batch** in each step of gradient descent
- Use only mini-batch B to compute

$$\nabla L = \frac{1}{|B|} \sum_{i \in B} \nabla L_i$$

- This **approximates** the real gradient

1: Stochastic Gradient Descent

Approximate down-hill steps but much faster to compute



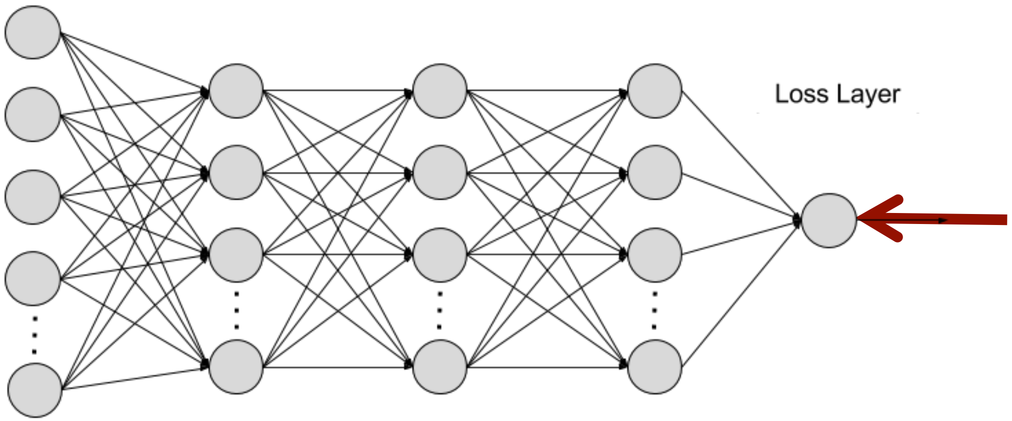
2: Computing the Gradient

- 2^{nd} trick: Compute ∇L_i step by step
- Neuron activations are chains of activation functions and matrix-vector multiplications
- Many connections between the neurons
- Computing this 109.386 dimensional gradient can be tricky...

backpropagation algorithm

2: Backpropagation

- The idea is to break down the gradient computation into smaller steps



- Key ingredients of backpropagation:

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

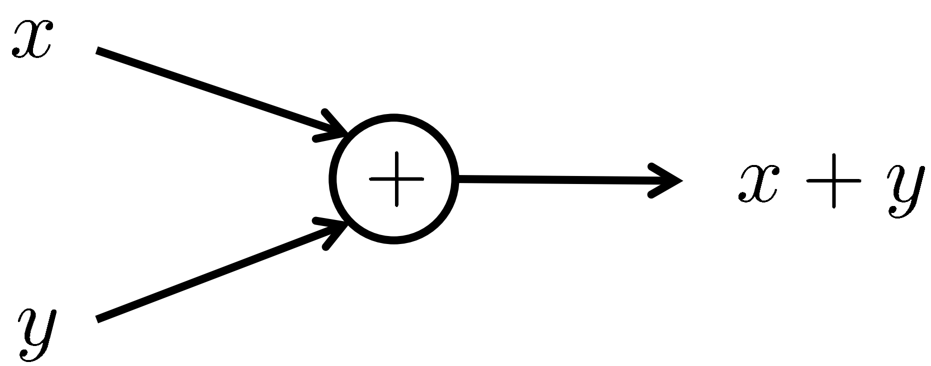
chain rule

local variables along paths through the NN

Backpropagation

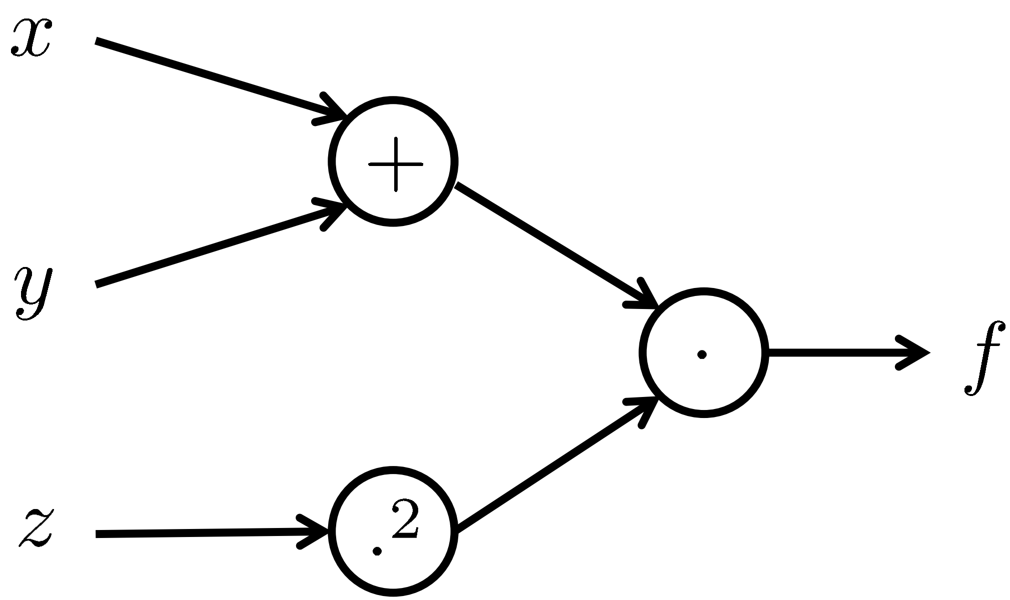
Computational Graph

- Directed graph
- Nodes contain mathematical operations
- Edges encode input/output values
- Example:



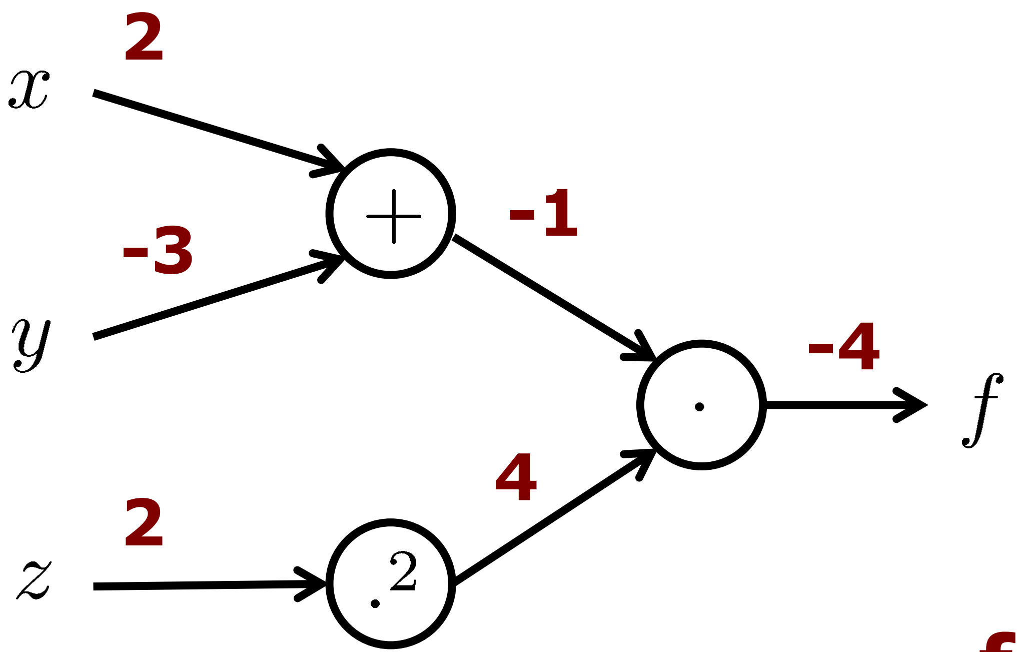
Function and Corresponding Computational Graph

$$f(x, y, z) = (x + y)\, z^2$$



Evaluating the Function

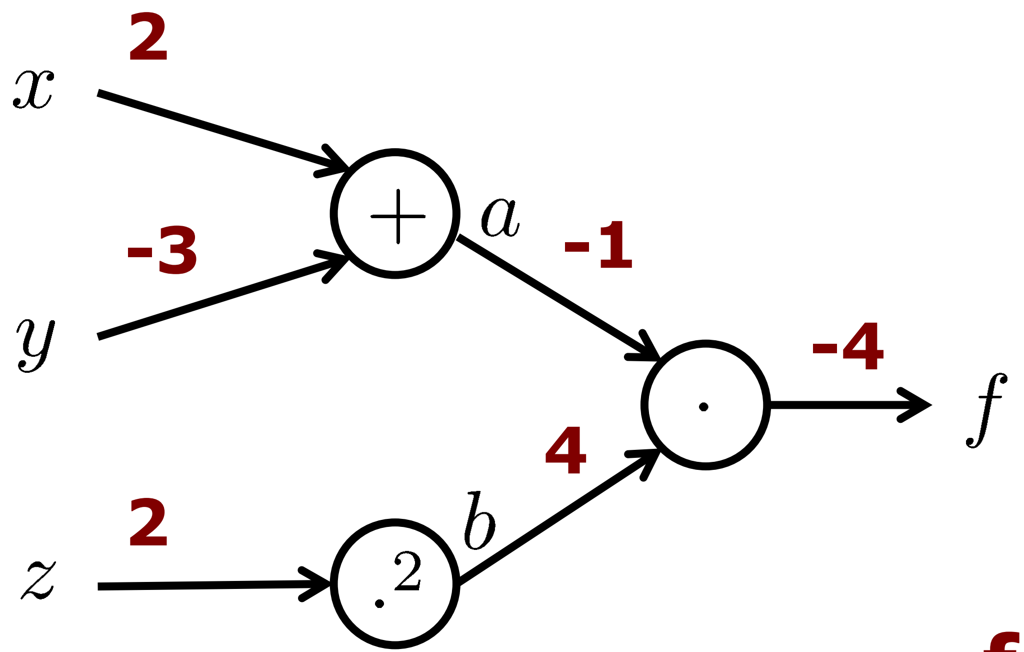
$$f(x, y, z) = (x + y)\, z^2 \qquad f(2, -3, 2) = -4$$



forward pass

Add Local Variables

$$f(x, y, z) = (x + y)\, z^2$$



$$a = x + y$$
$$b = z^2$$
$$f = a\, b$$

forward pass

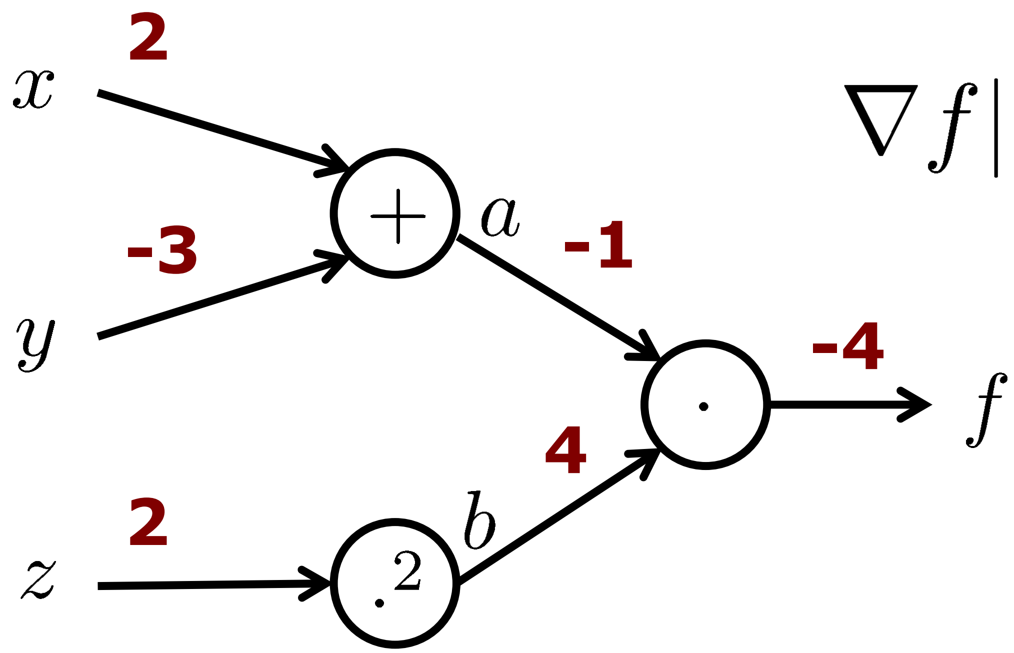
Computing the Gradient

- With the forward pass, we evaluate the function at a given point
- Next: compute the gradient of the function at the given point
- We can do this by traversing the graph backwards
- At each note, we compute the local derivative of the function w.r.t. the local inputs

Gradient At Point [2, -3, 2]

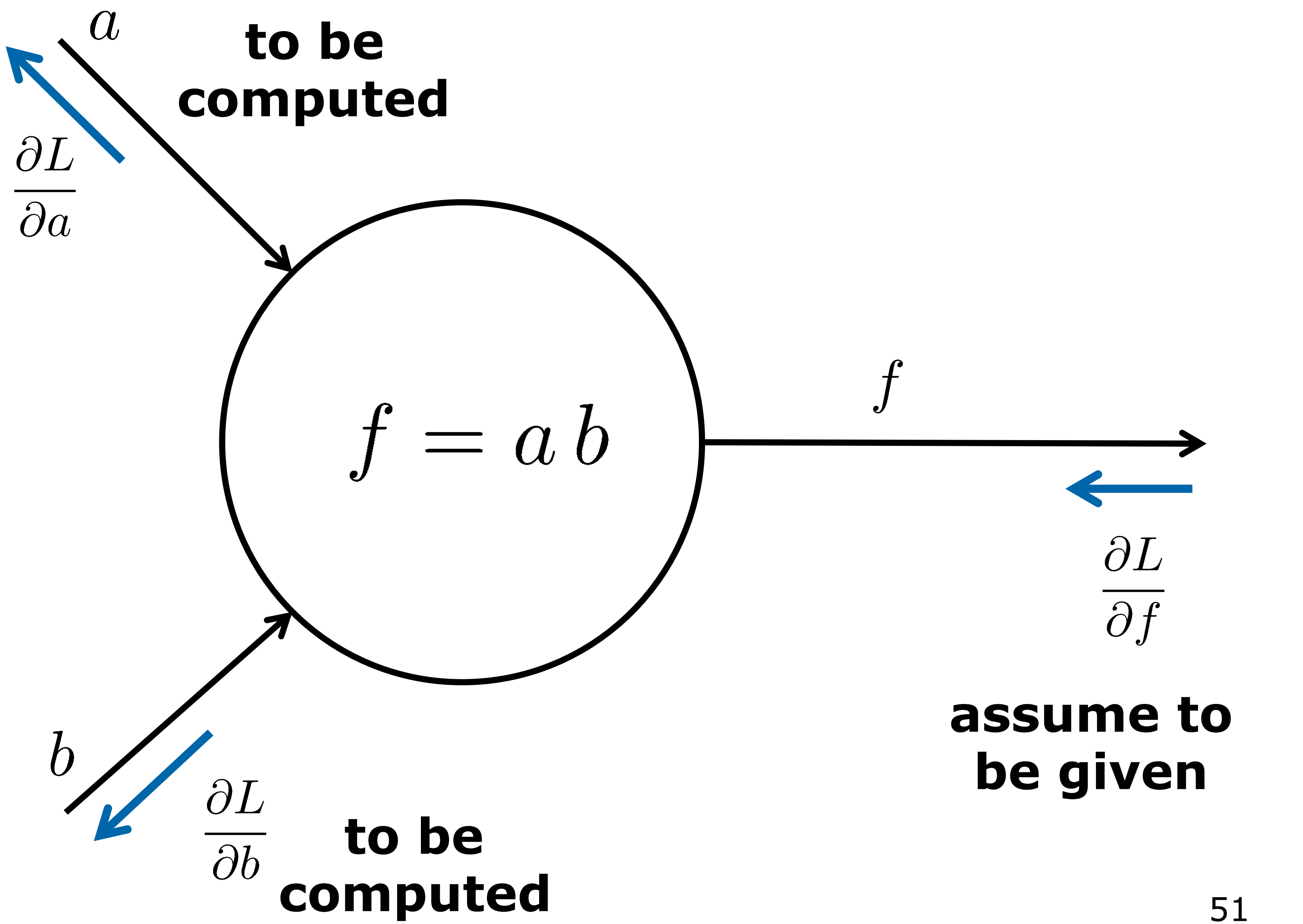
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right]^\top$$

$$\nabla f \Big|_{[2,-3,2]^\top} = ?$$

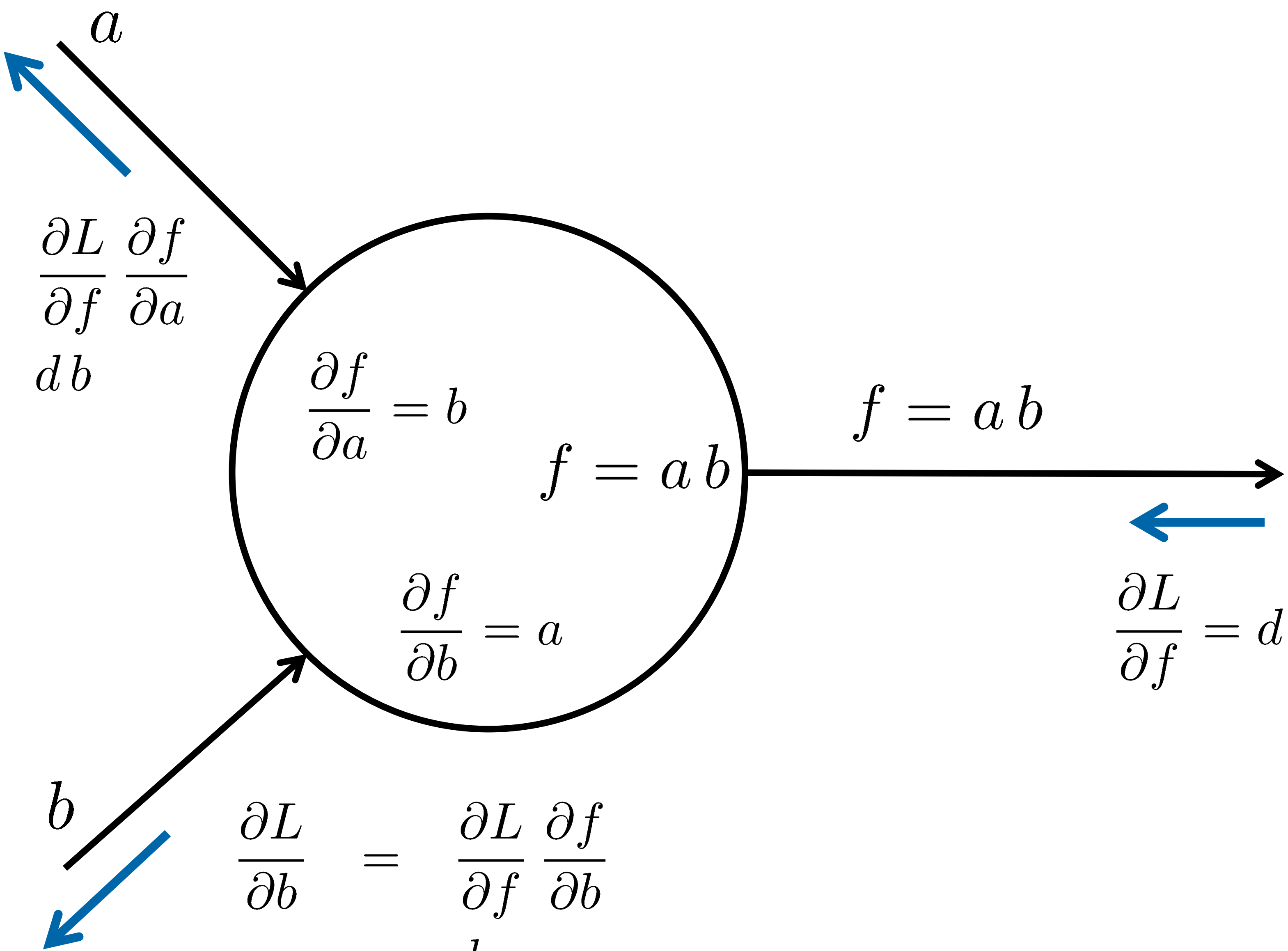


backward pass

For a Single Node (Chain Rule)



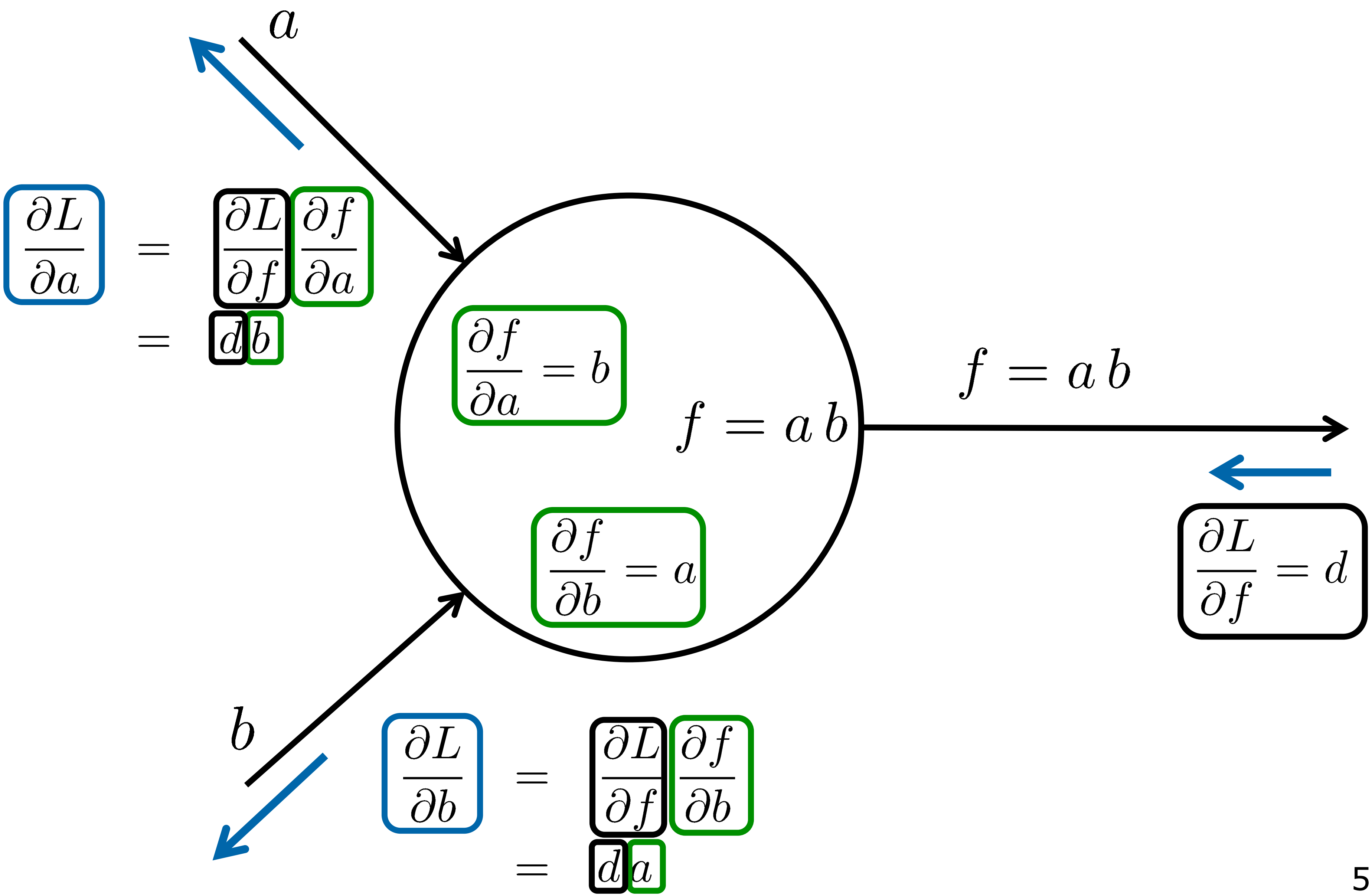
For a Single Node (Chain Rule)



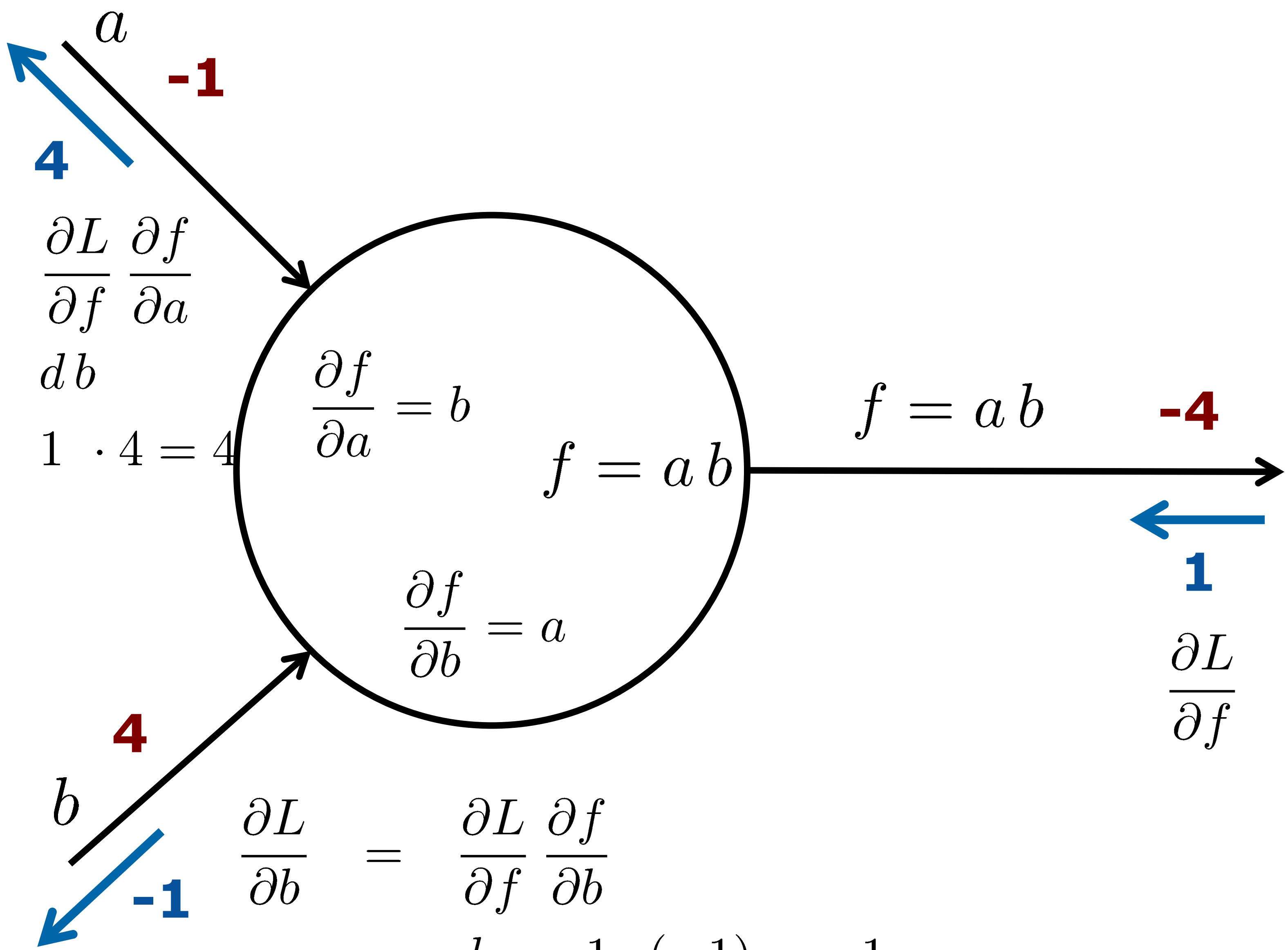
$$\begin{aligned}\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial f}\,\frac{\partial f}{\partial a} \\ &= d\, b\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial f}\,\frac{\partial f}{\partial b} \\ &= d\, a\end{aligned}$$

For a Single Node (Chain Rule)



For a Single Node (Chain Rule)



$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial a}$$

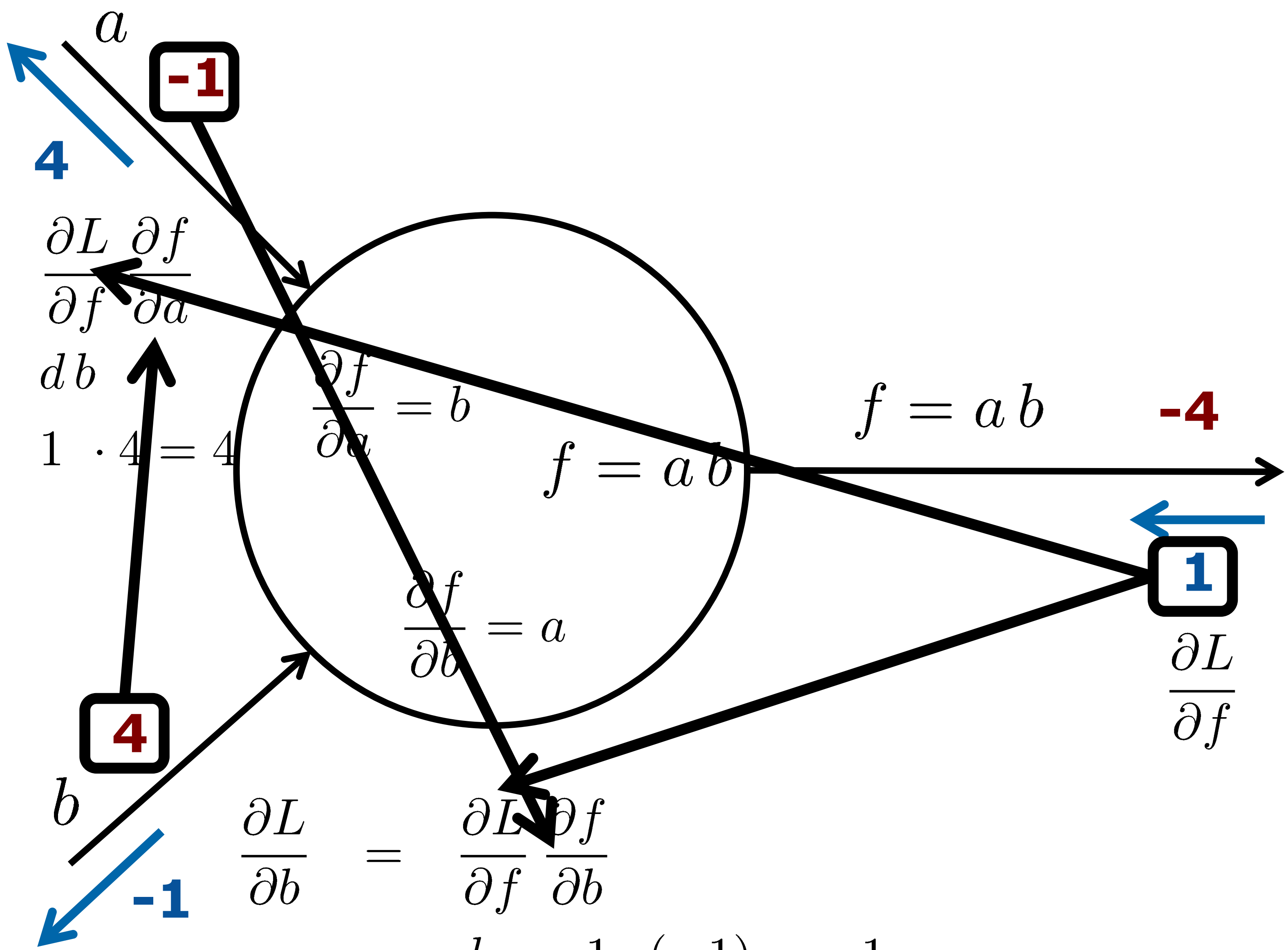
$$= d\,b$$

$$= 1 \cdot 4 = 4$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial b}$$

$$= d\,a = 1 \cdot (-1) = -1$$

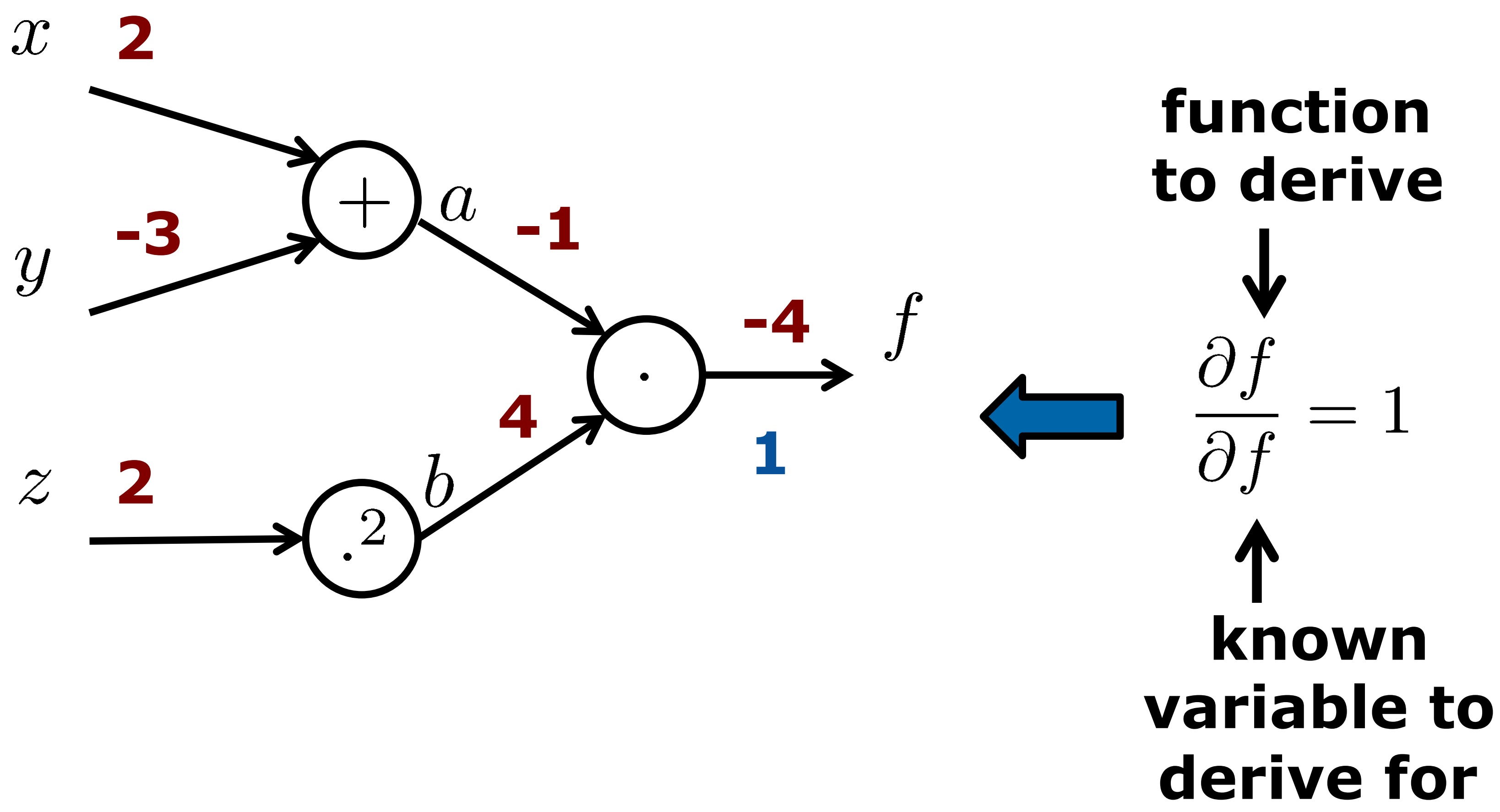
For a Single Node (Chain Rule)



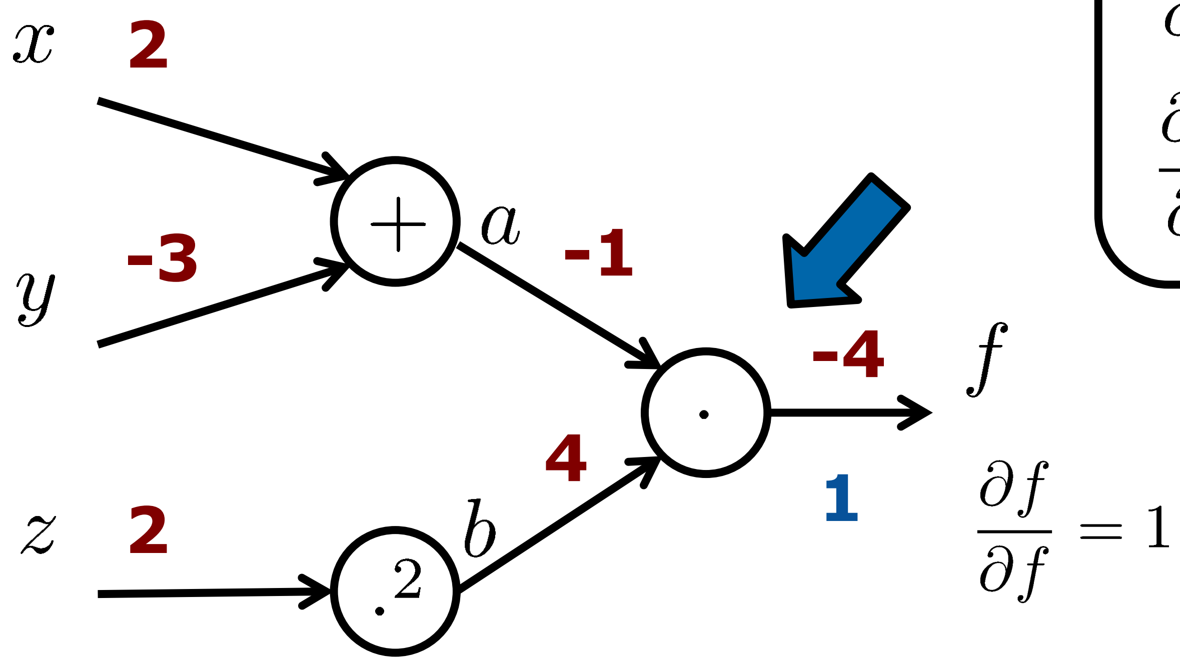
$$
\begin{aligned}
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial f}\frac{\partial f}{\partial a} \\
&= d\,b \\
&= 1 \cdot 4 = 4
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial f}\frac{\partial f}{\partial b} \\
&= d\,a = 1 \cdot (-1) = -1
\end{aligned}
$$

Backward Pass



Backward Pass



function

$$f = a\,b$$

$$\frac{\partial f}{\partial a} = b$$

$$\frac{\partial f}{\partial b} = a$$

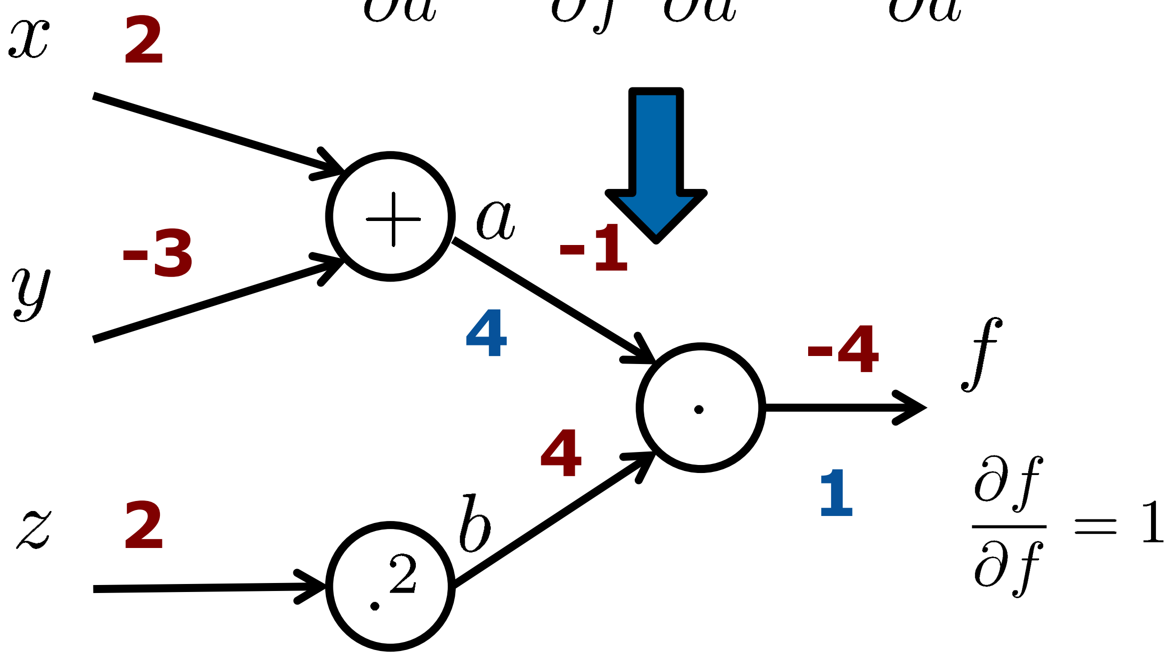
Backward Pass

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial a} = 1\frac{\partial f}{\partial a} = b = 4$$

function

$$f = a\,b$$

$$\frac{\partial f}{\partial a} = b$$

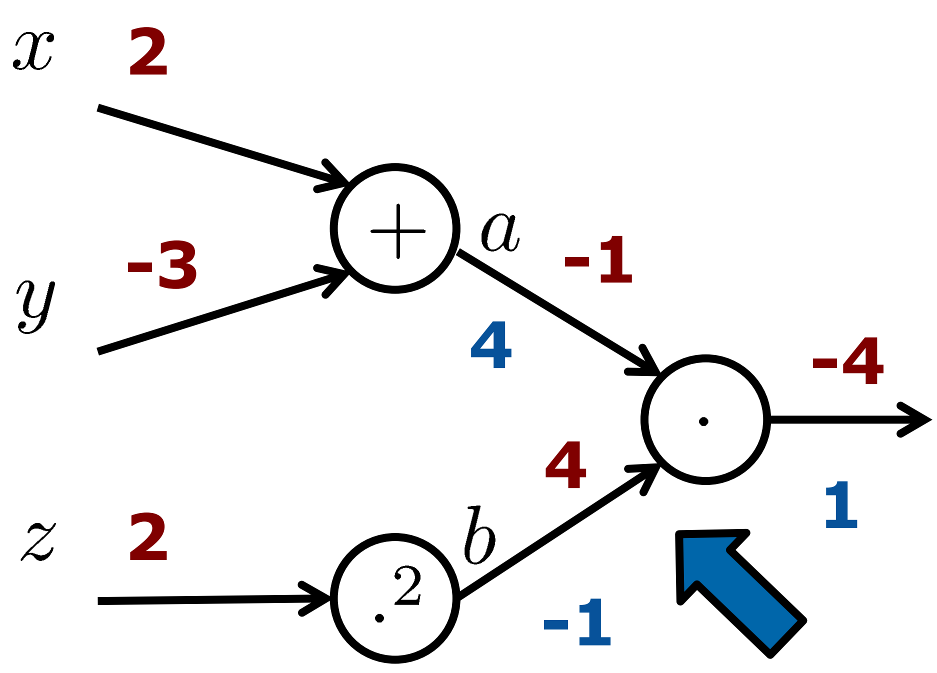


Backward Pass

function

$$f = a\,b$$

$$\frac{\partial f}{\partial b} = a$$

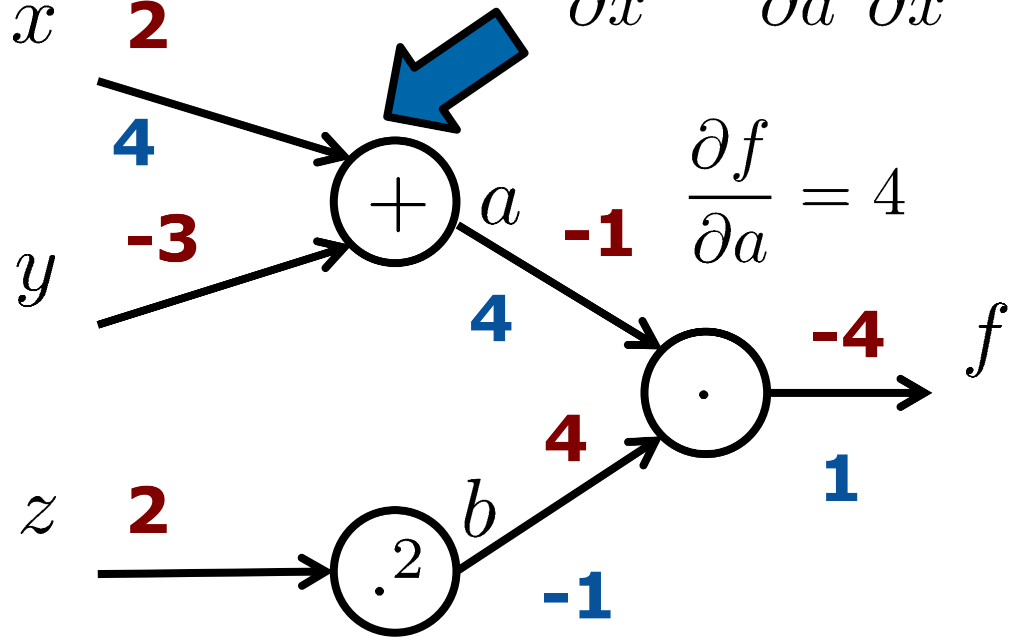


$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial f}\,\frac{\partial f}{\partial b} = 1\,\frac{\partial f}{\partial b} = a = -1$$

Backward Pass

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} = 4\,\frac{\partial a}{\partial x} = 4$$



function

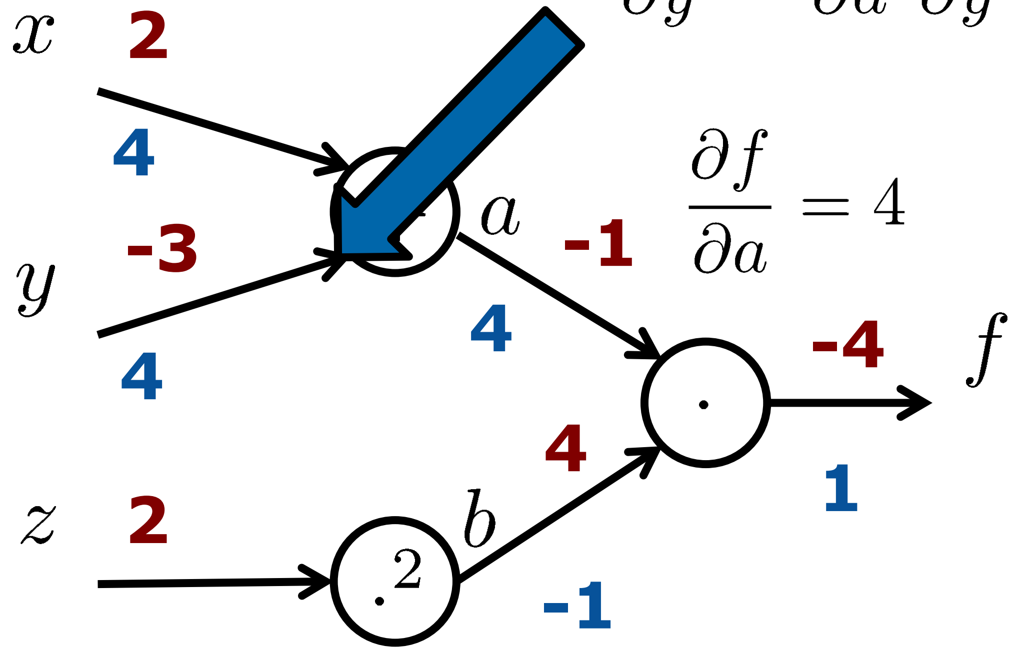
$$a = x + y$$

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

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

Backward Pass

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\,\frac{\partial a}{\partial y} = 4\frac{\partial a}{\partial y}\,4 = 4$$



$$\frac{\partial f}{\partial a} = 4$$

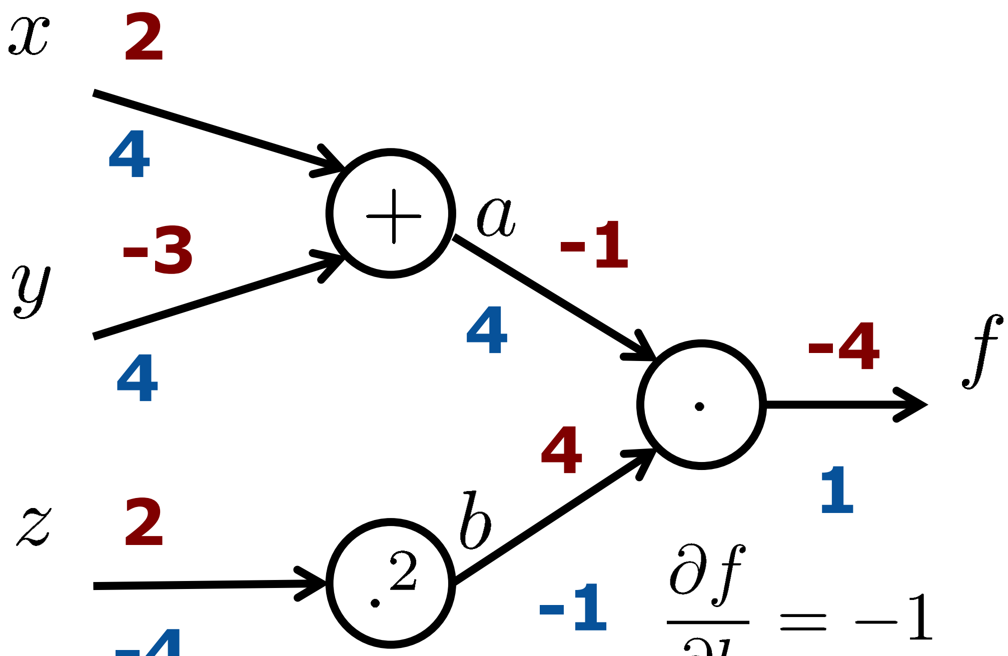
function

$$a = x + y$$

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

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

Backward Pass



function

$$b = z^2$$

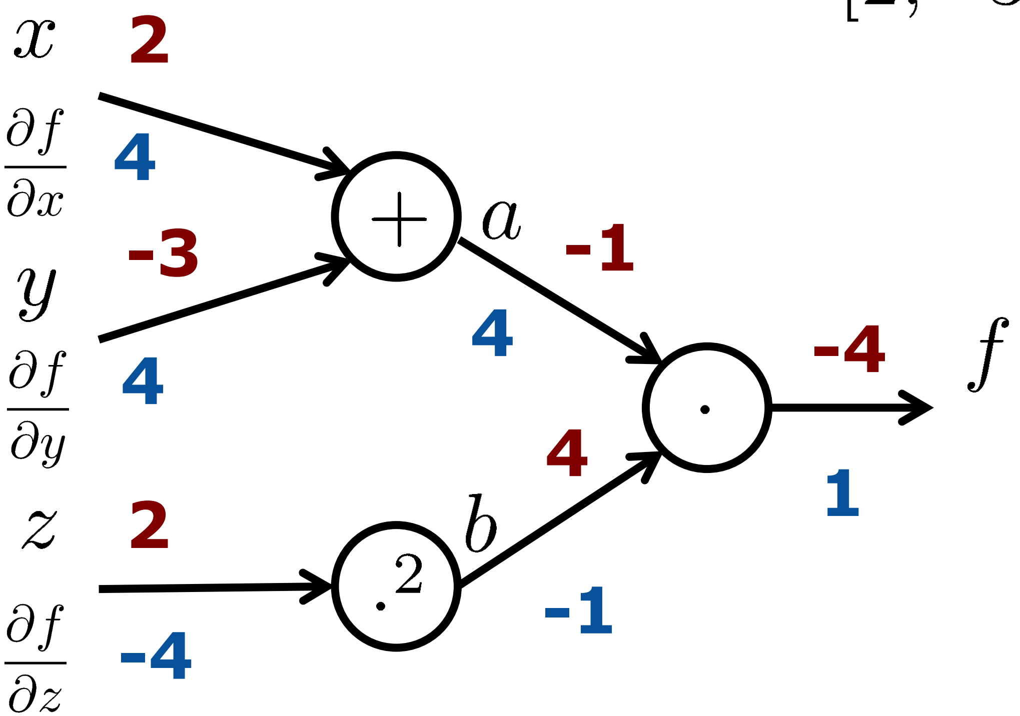
$$\frac{\partial b}{\partial z} = 2z$$

$$\frac{\partial f}{\partial b} = -1$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial z} = (-1) \cdot \frac{\partial b}{\partial z} = -2z = -4$$

Backward Pass

$$\nabla f \big|_{[2,-3,2]^\top} = \begin{bmatrix} 4 \\ 4 \\ -4 \end{bmatrix}$$



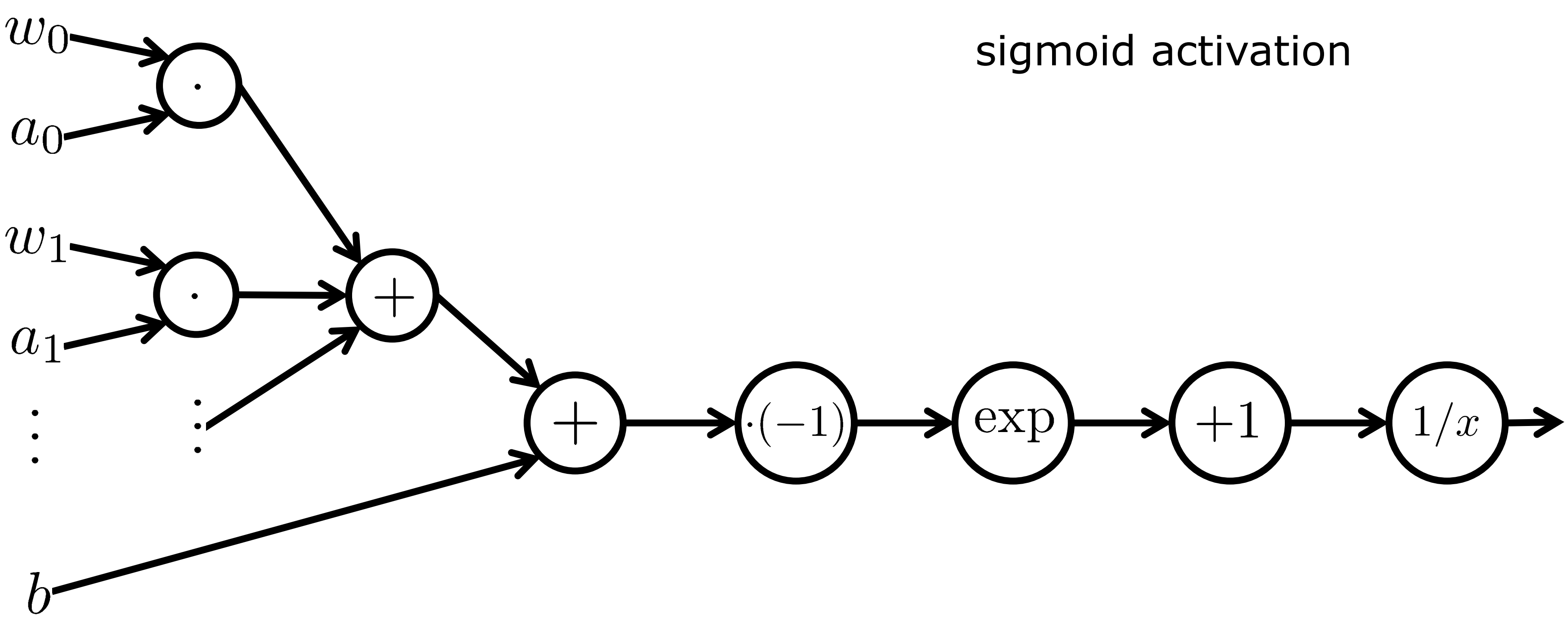
Backpropagation (Backprop)

- Approach is called **backpropagation** and computes the gradient of any function expressed as such a graph
- Combines **chain rule** and local variables for the graph nodes
- Recursively traverses the graph
- Can be used for computing both, numerical and analytical gradient computation

Example for One Neuron

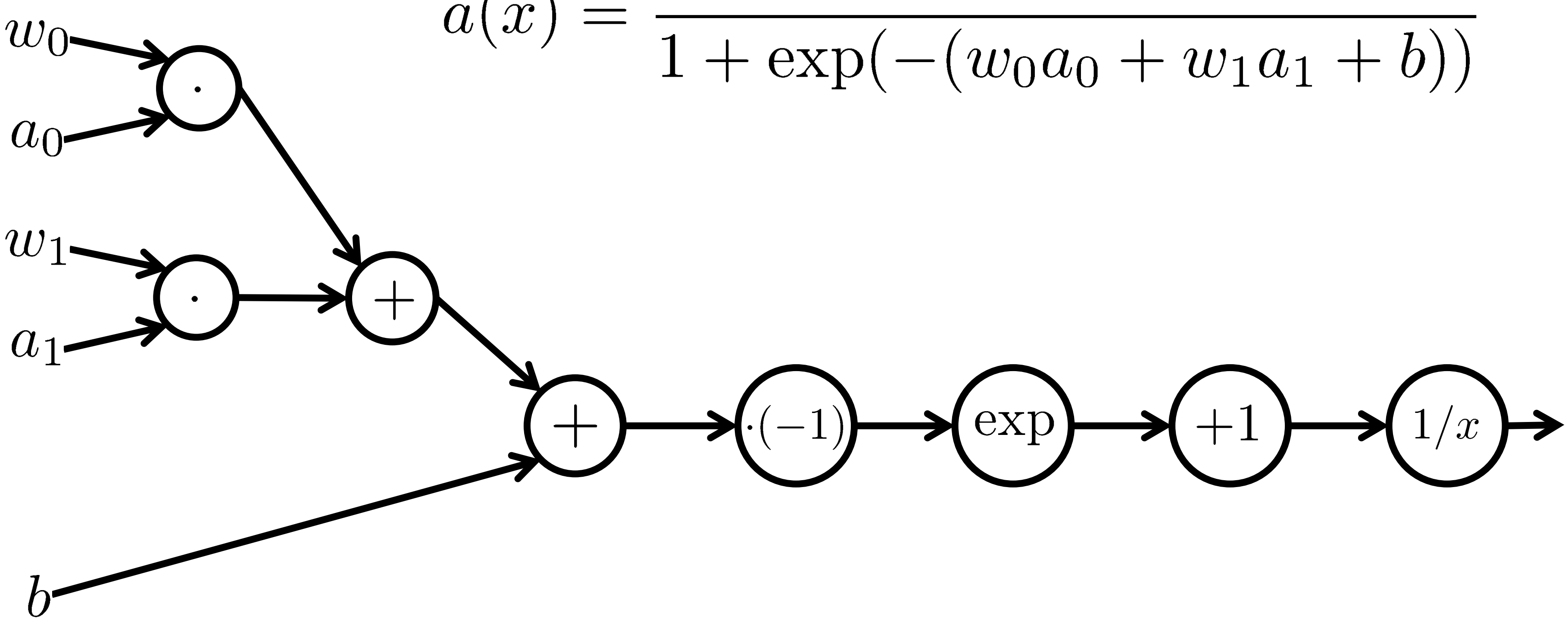
$$a = \sigma\left(\sum w_n a_n + b\right) \qquad \sigma(x) = \frac{1}{1+\exp(-x)}$$

sigmoid activation



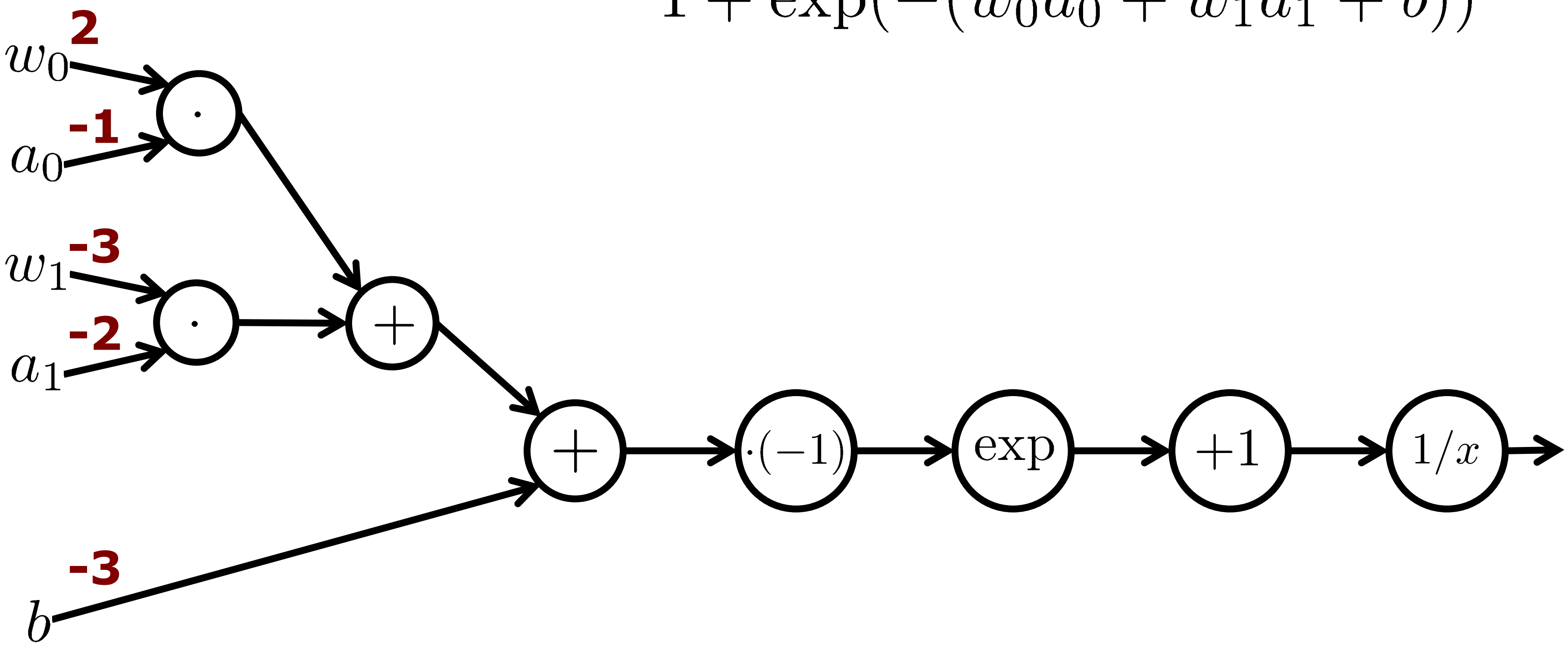
Example for One Neuron with Two Incoming Activations

$$a(x) = \frac{1}{1 + \exp(-(w_0 a_0 + w_1 a_1 + b))}$$



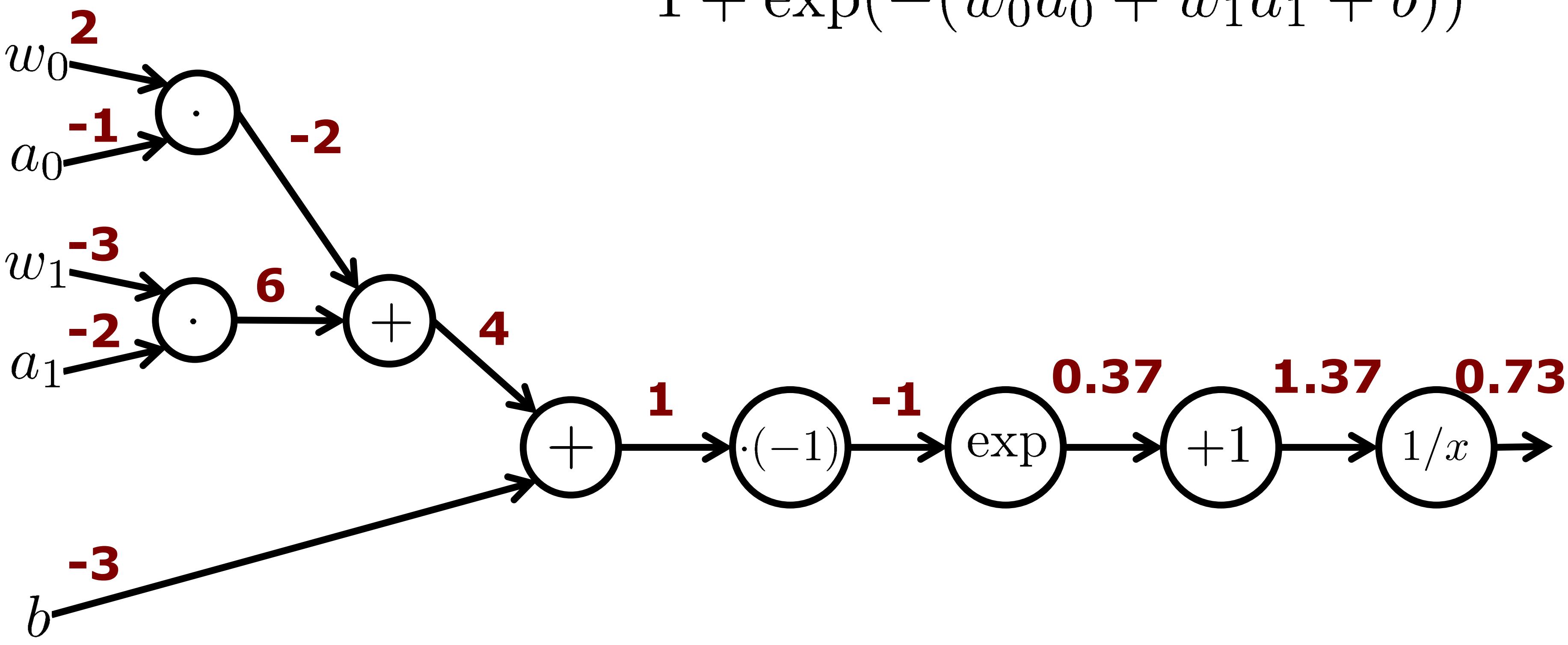
Forward Pass

$$a(x) = \frac{1}{1 + \exp(-(w_0 a_0 + w_1 a_1 + b))}$$

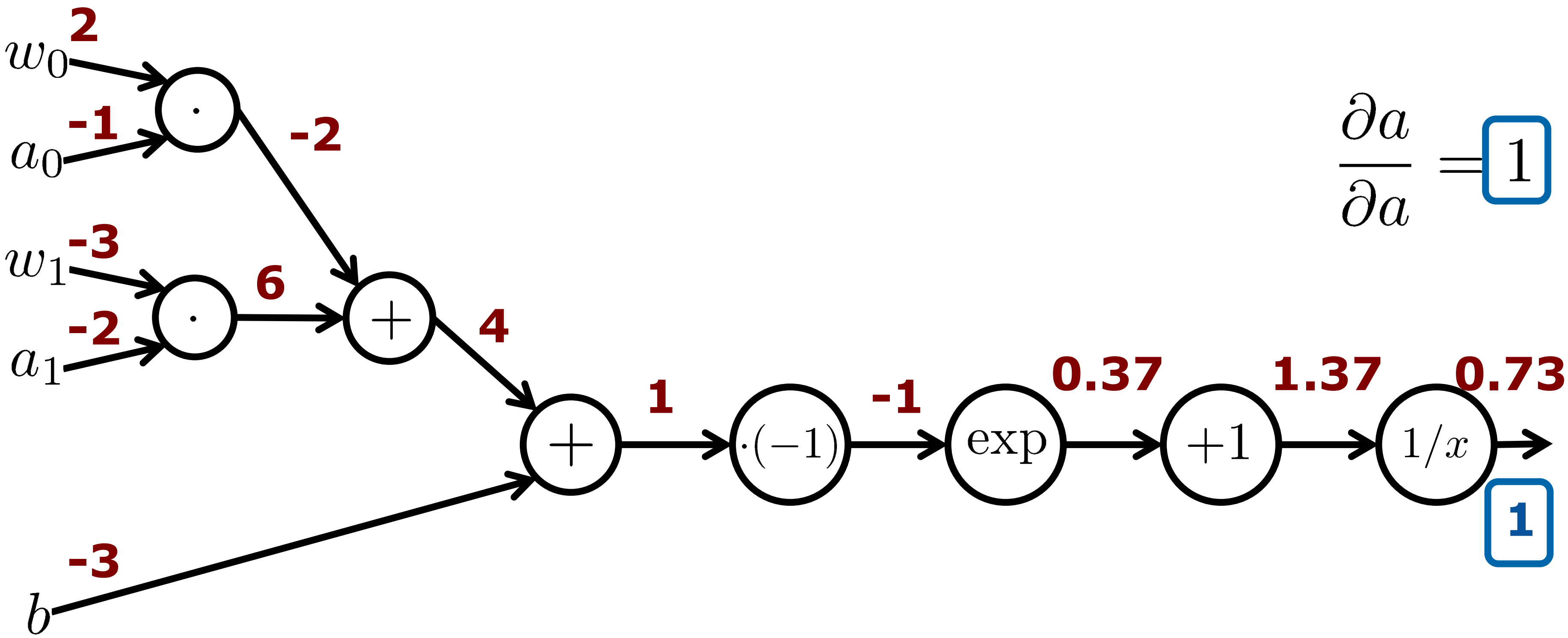


Forward Pass

$$a(x) = \frac{1}{1 + \exp(-(w_0 a_0 + w_1 a_1 + b))}$$

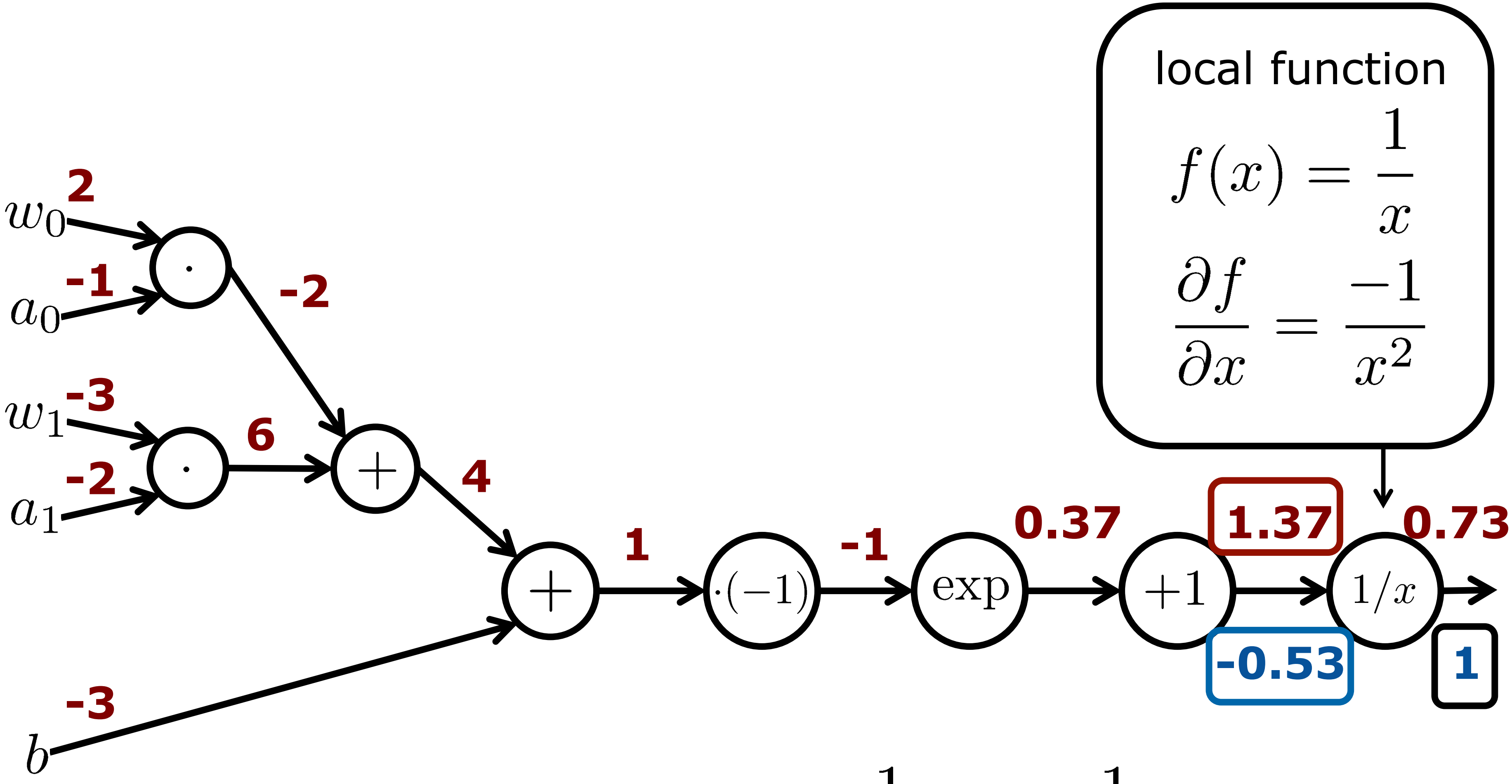


Backward Pass



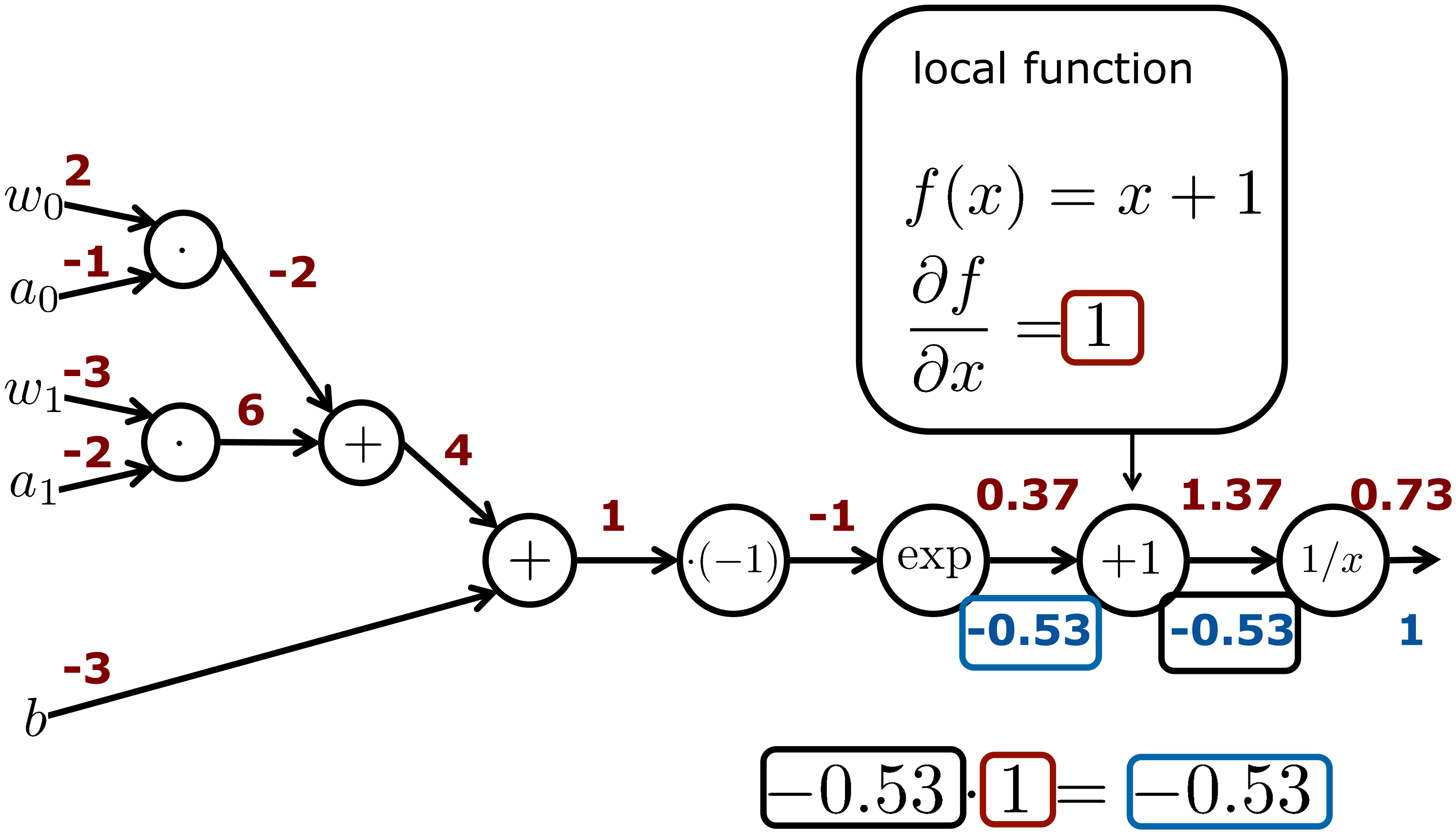
$$\frac{\partial a}{\partial a} = \boxed{1}$$

Backward Pass

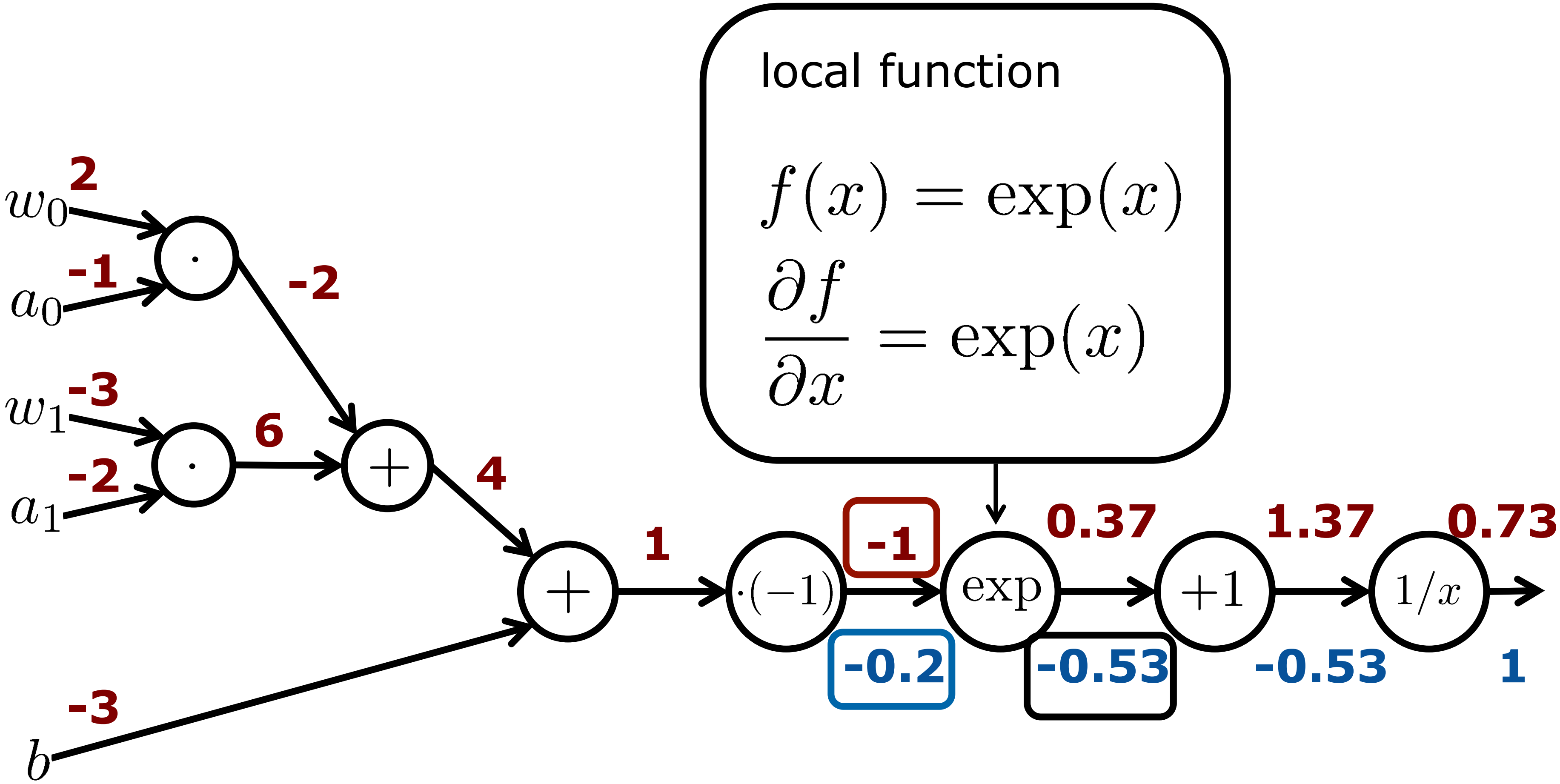


$$\boxed{1} \cdot \frac{-1}{x^2} = \frac{-1}{\boxed{1.37}^2} = \boxed{-0.53}$$

Backward Pass

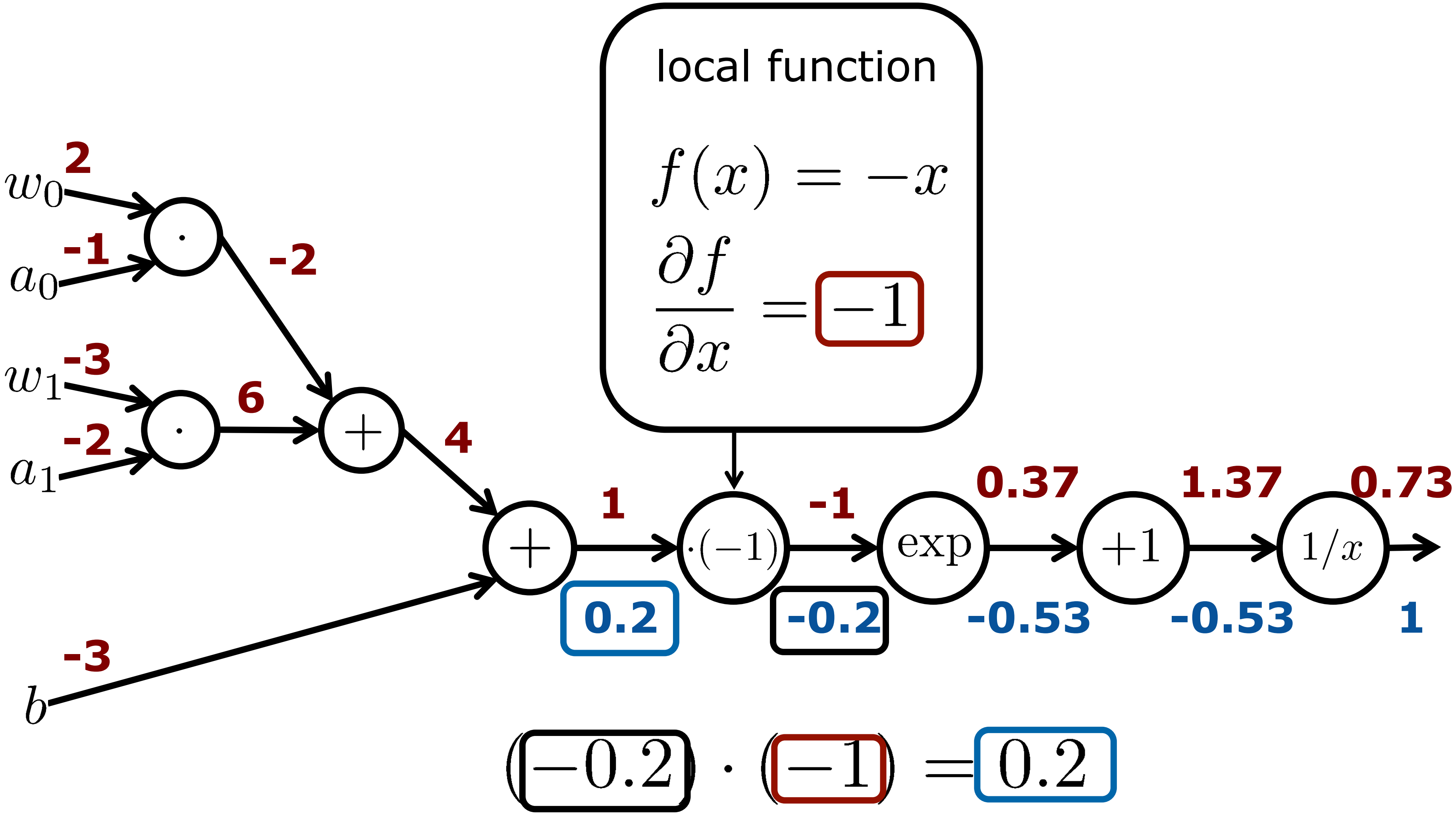


Backward Pass

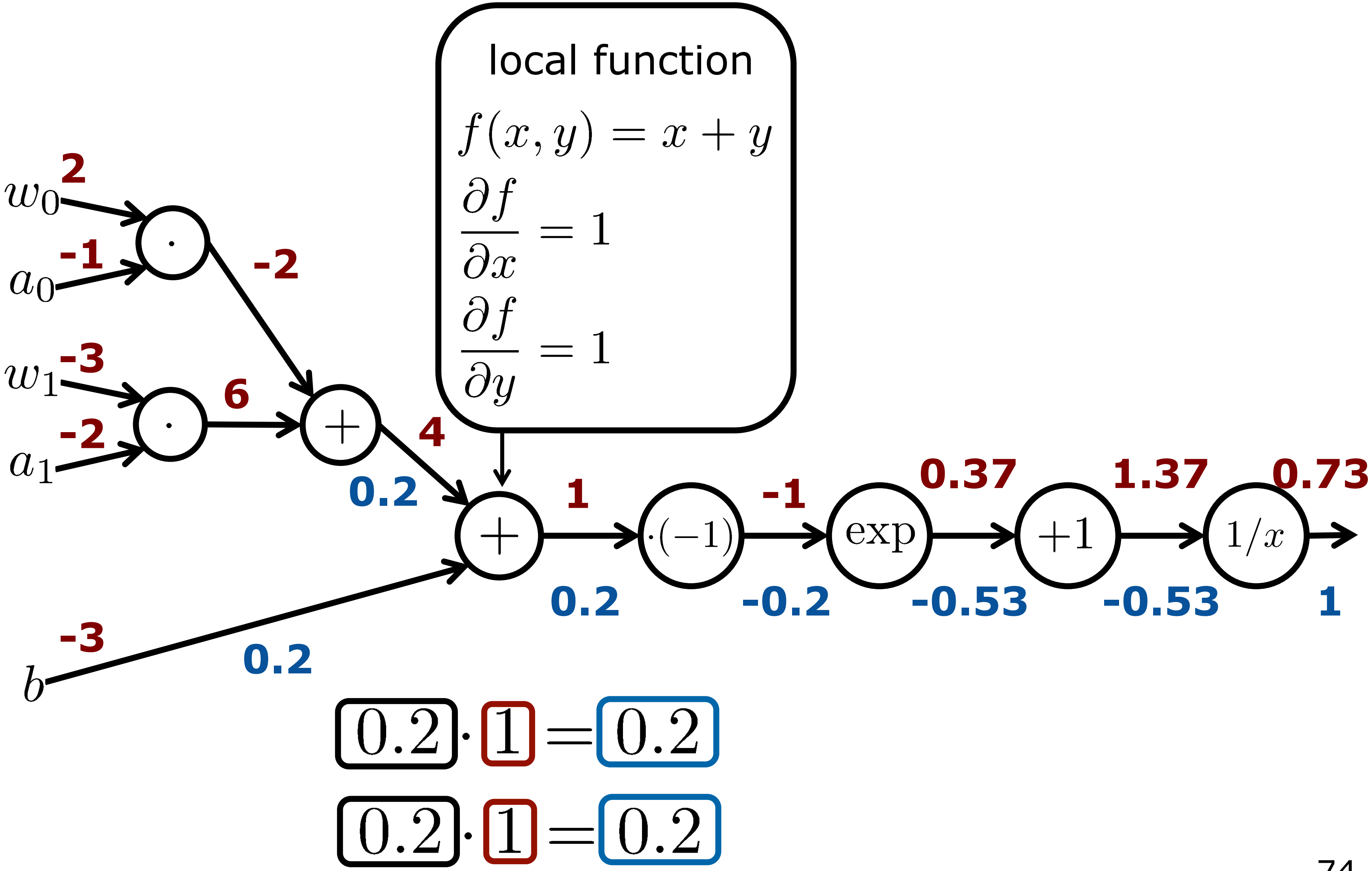


$$\boxed{-0.53} \cdot \exp(\boxed{x}) = -0.53 \cdot \exp(-1) = \boxed{-0.2}$$

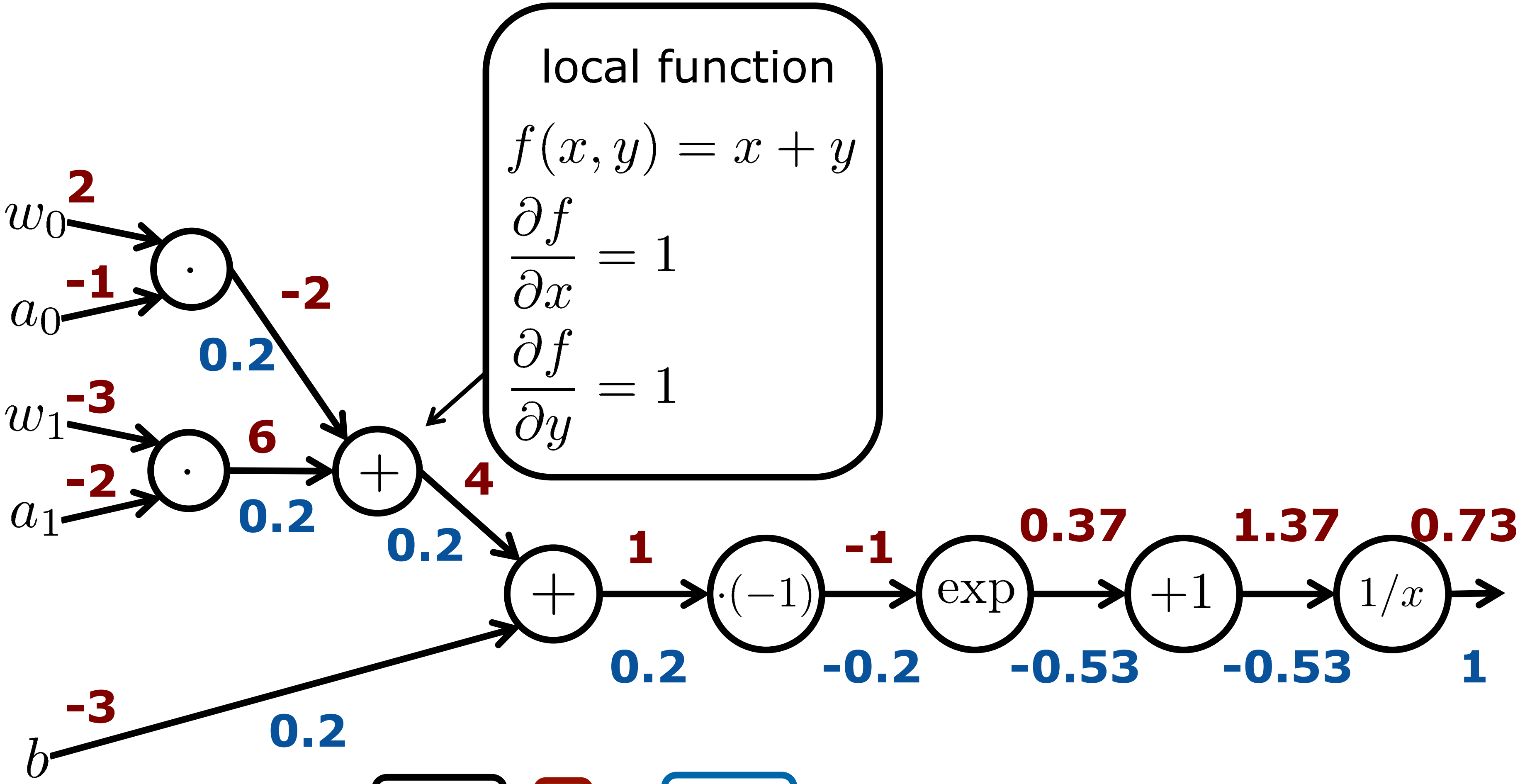
Backward Pass



Backward Pass



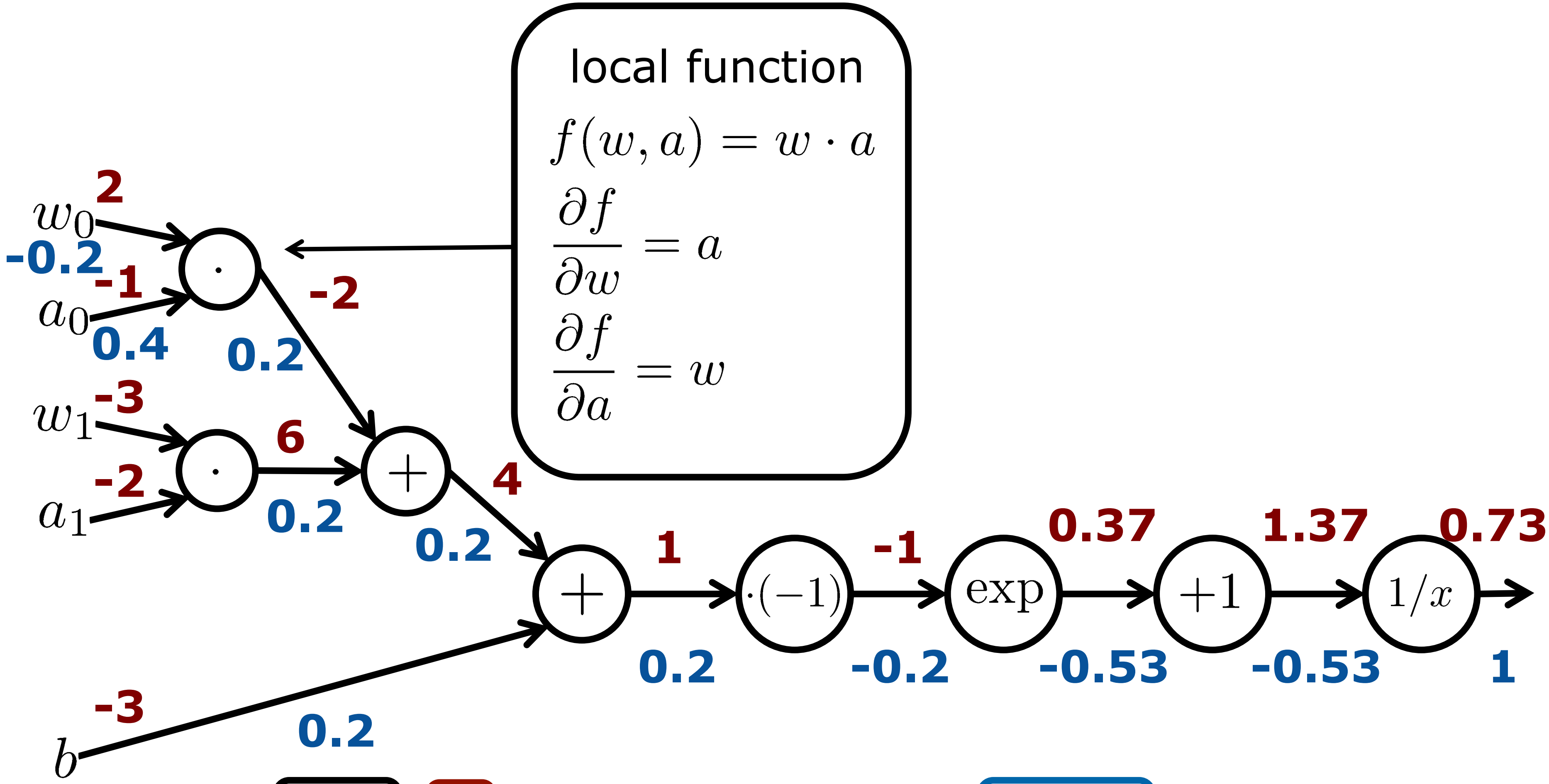
Backward Pass



$$\boxed{0.2} \cdot \boxed{1} = \boxed{0.2}$$

$$\boxed{0.2} \cdot \boxed{1} = \boxed{0.2}$$

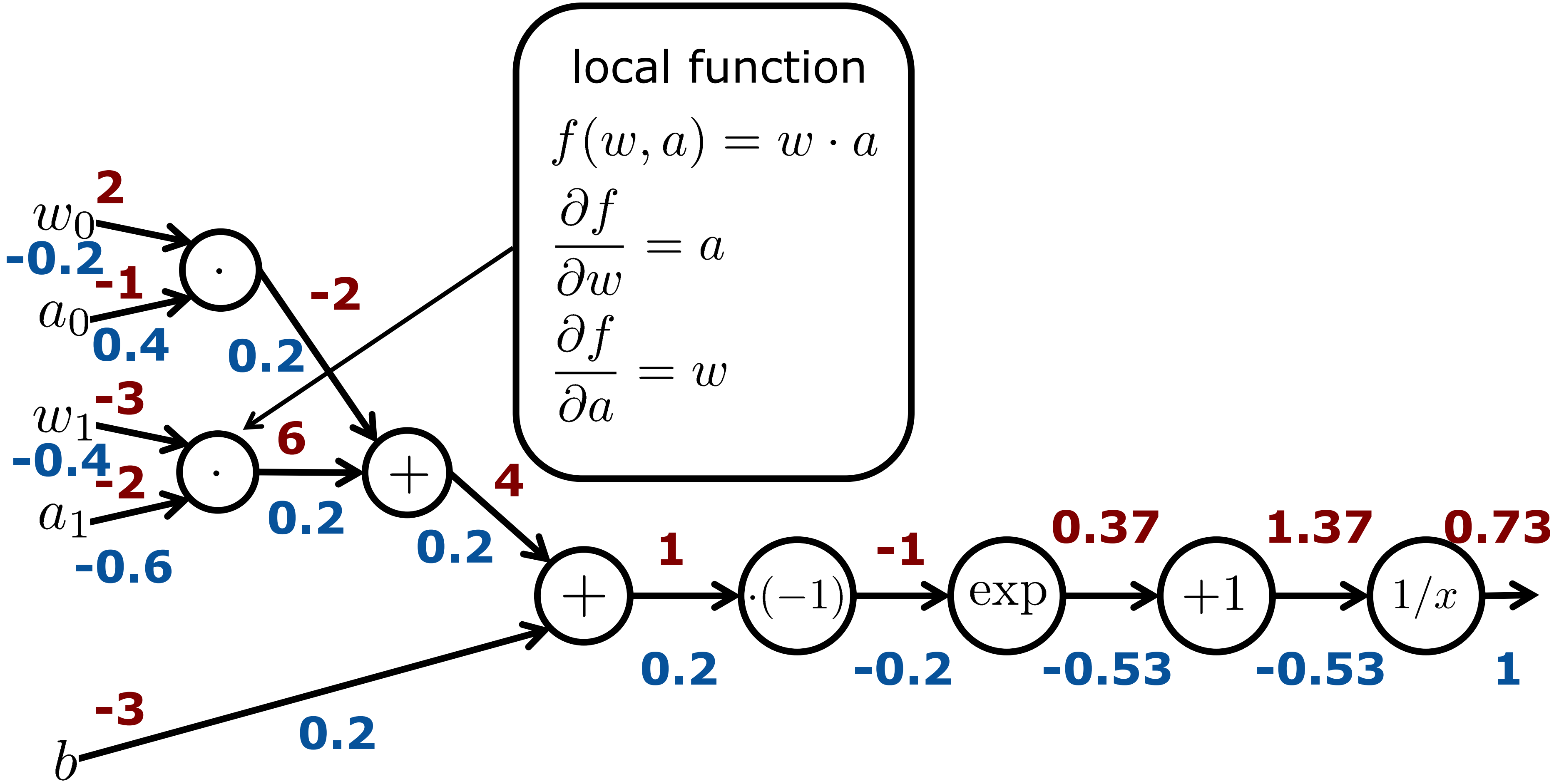
Backward Pass



$$\boxed{0.2} \cdot \boxed{a} = 0.2 \cdot (-1) = \boxed{-0.2}$$

$$\boxed{0.2} \cdot \boxed{w} = 0.2 \cdot 2 = \boxed{0.4}$$

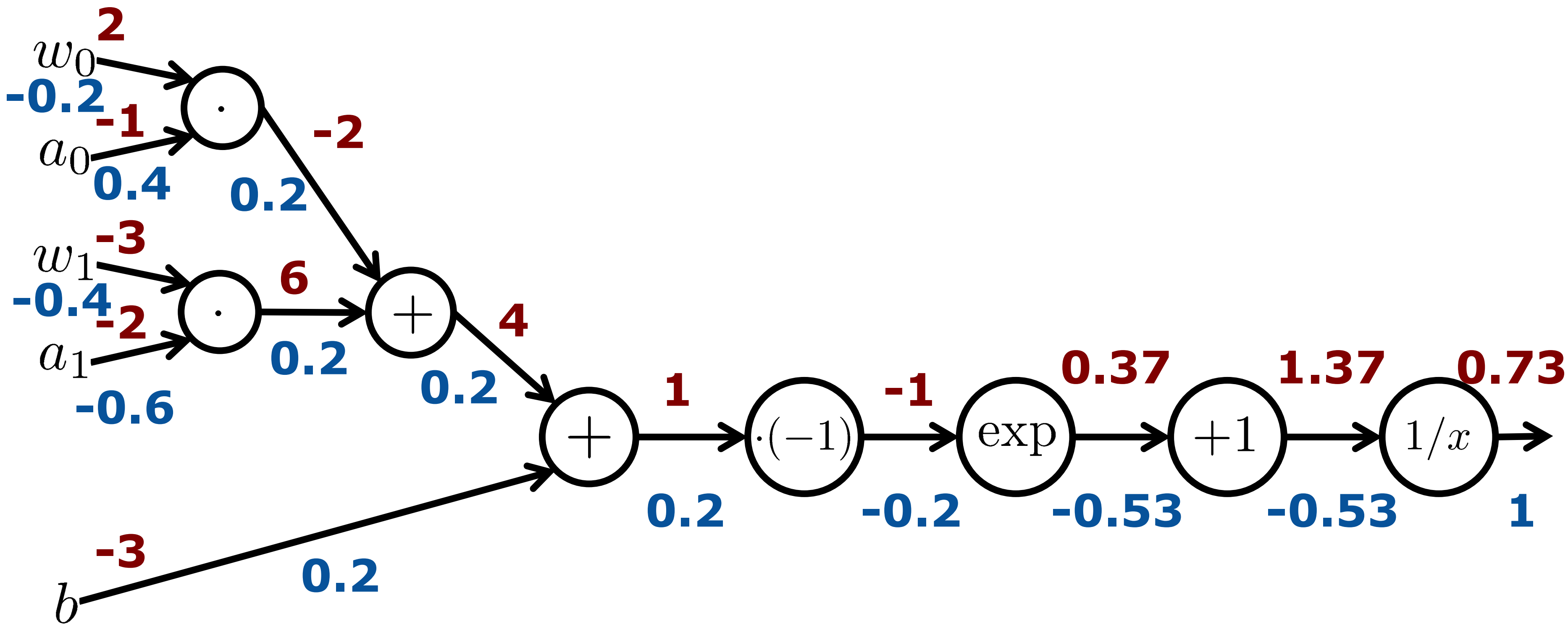
Backward Pass



$$0.2 \cdot a = 0.2 \cdot (-2) = -0.4$$

$$0.2 \cdot w = 0.2 \cdot (-3) = -0.6$$

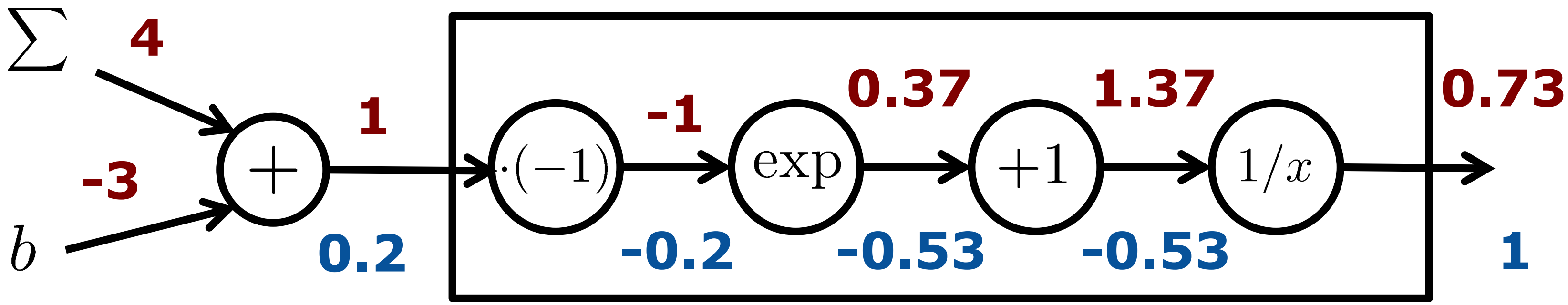
Backward Pass



$$\nabla\sigma\big|_{[2,-1,-3,-2,-3]^\top} = [-0, 2, 0.4, -0.4, -0.6, 0.2]^\top$$

Backpropagation for a Single Neuron with Sigmoid Activ.

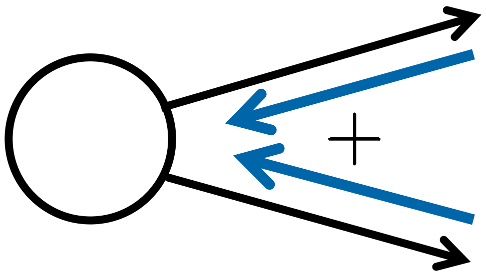
- The example illustrated that we can compute the gradient for a neuron
- We can also model the sigmoid using a single node ("sigmoid gate")



$$\sigma(x) = \frac{1}{1+\exp(-x)} \qquad \frac{\partial \sigma(x)}{\partial x} = (1-\sigma(x)) \cdot \sigma(x)$$

Backpropagation from Neuron to Neuron

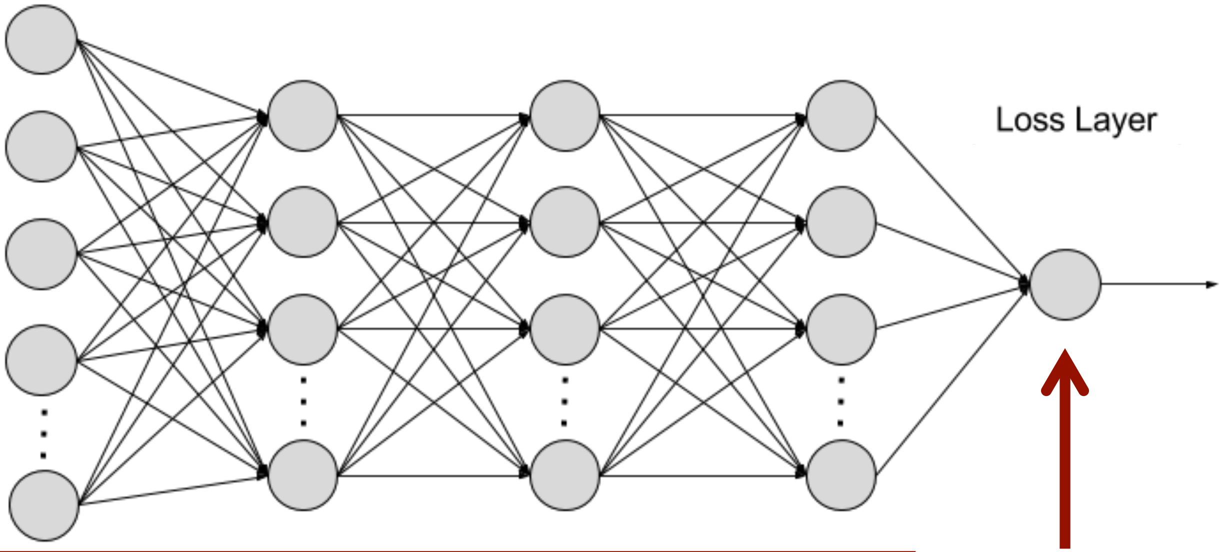
- Recursive application from neuron to neuron through the layers
- For multiple outgoing edges, we need to compute the sum of gradients



- Backpropagation also generalizes directly to multivariate function

Neural Networks As Computation Graphs

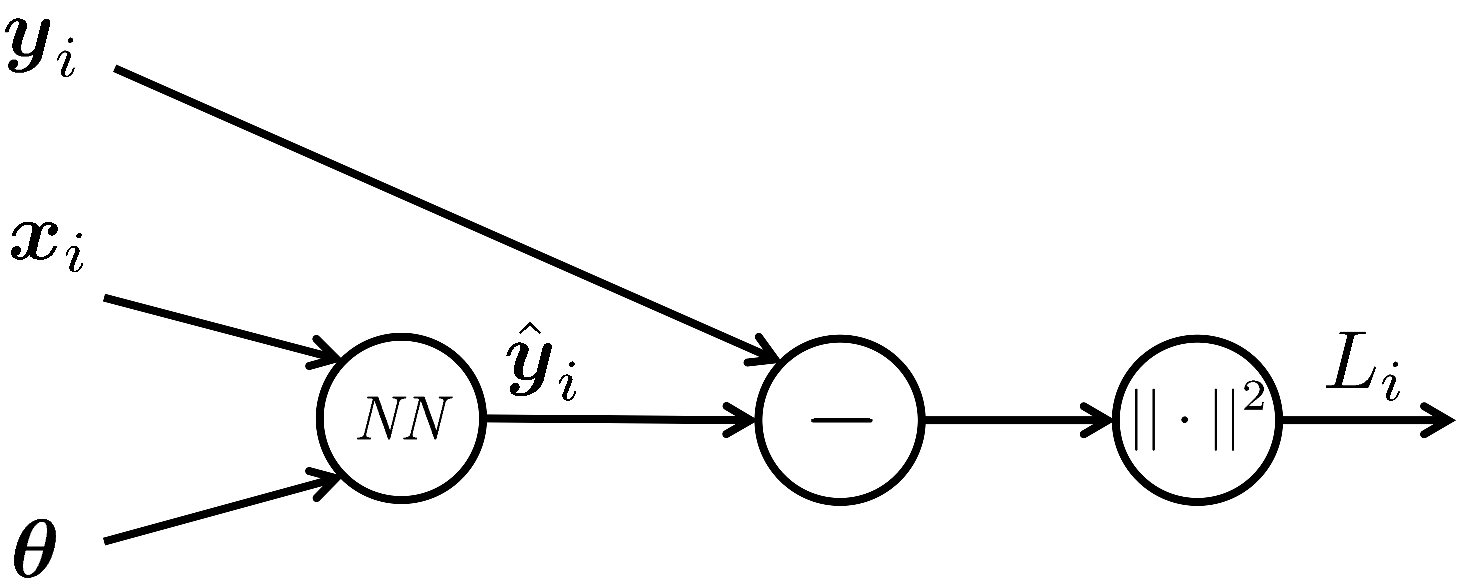
- We can use BP for computing the gradient of the loss function $L(\boldsymbol{\theta})$
- We add a loss layer



neural network **loss layer**

Quadratic Loss for Backpropagation

training data



parameters

Backpropagation (Backprop)

- BP allows us to compute gradients of nested and complex functions
- Combines chain rule and local variables
- Recursive traversal of the network
- Forward pass computes the linearization point for the gradient
- Backward pass computes the gradient

Learning a Neural Network

Repeat until convergence

1. Sample mini-batch from training data
2. Run backprop to compute gradient for SGD using mini-batch

$$\nabla L|_{\boldsymbol{\theta}^{(j)}}$$

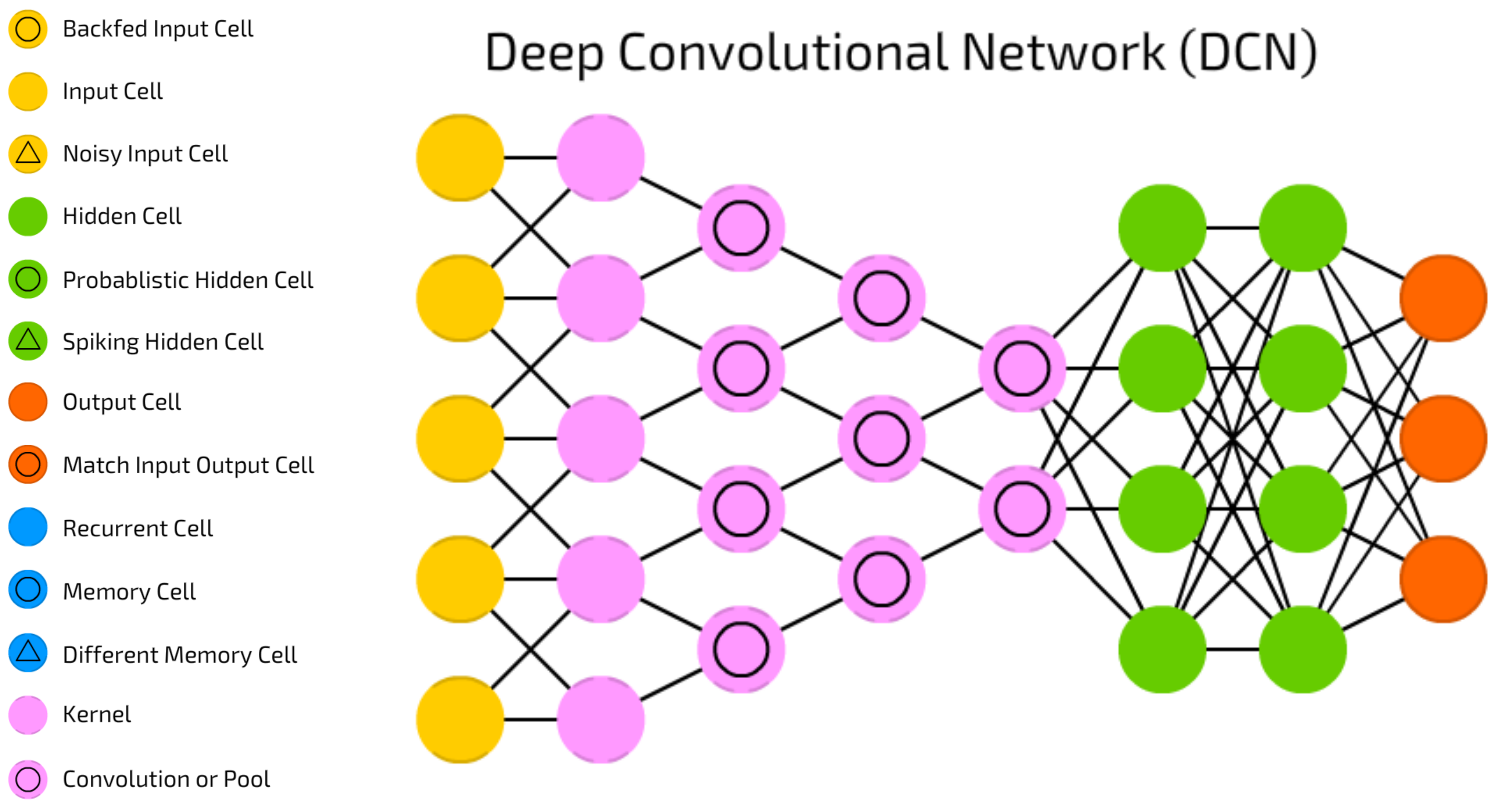
3. Execute SGD step to find better parameters reducing the loss

$$\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} - \lambda \, \nabla L|_{\boldsymbol{\theta}^{(j)}}$$

Return parameter vector

Convolutional Neural Networks

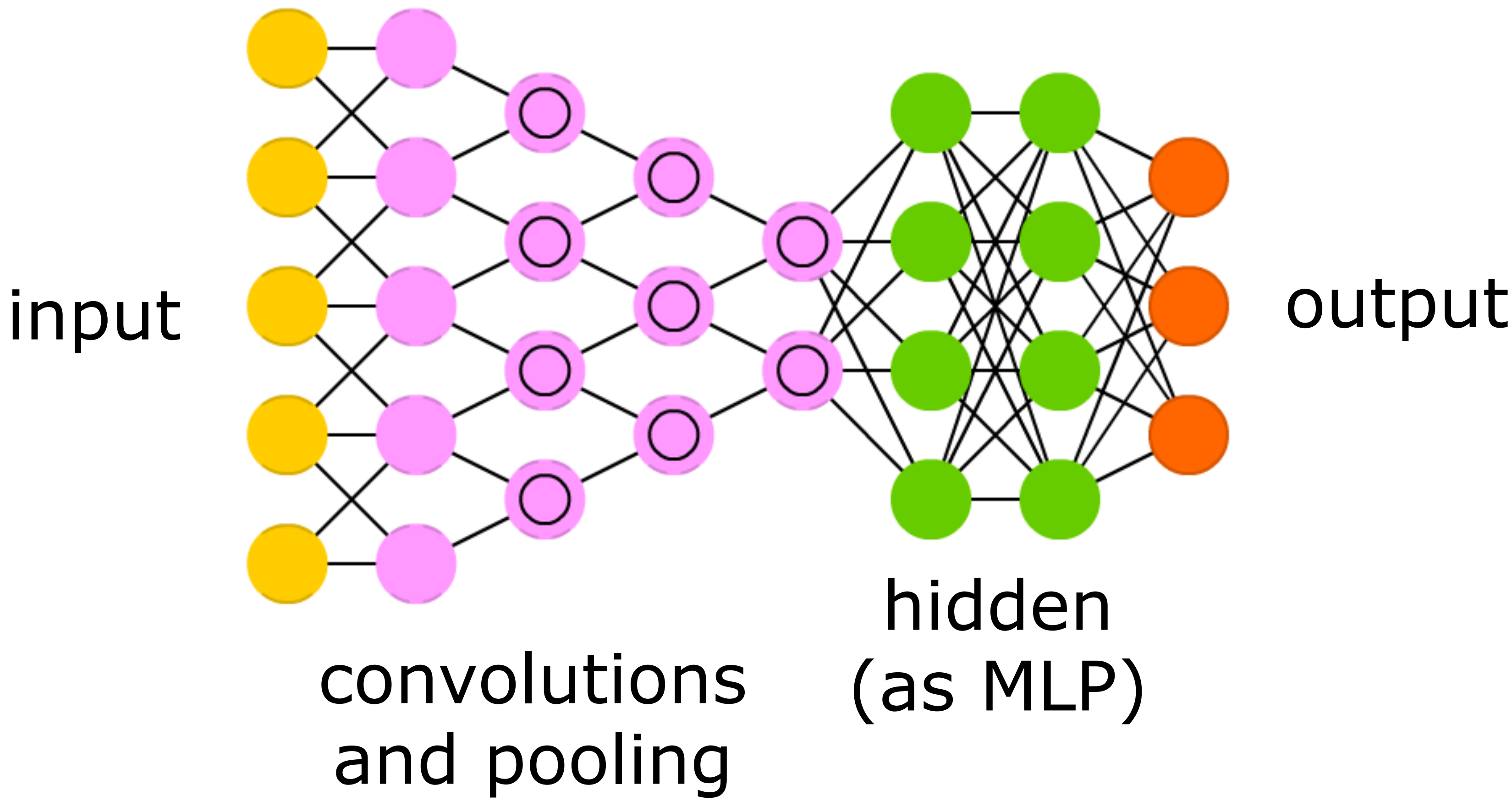
In image-related learning tasks, CNNs play an important role



[Image courtesy: van Veen]

Convolutional Neural Networks

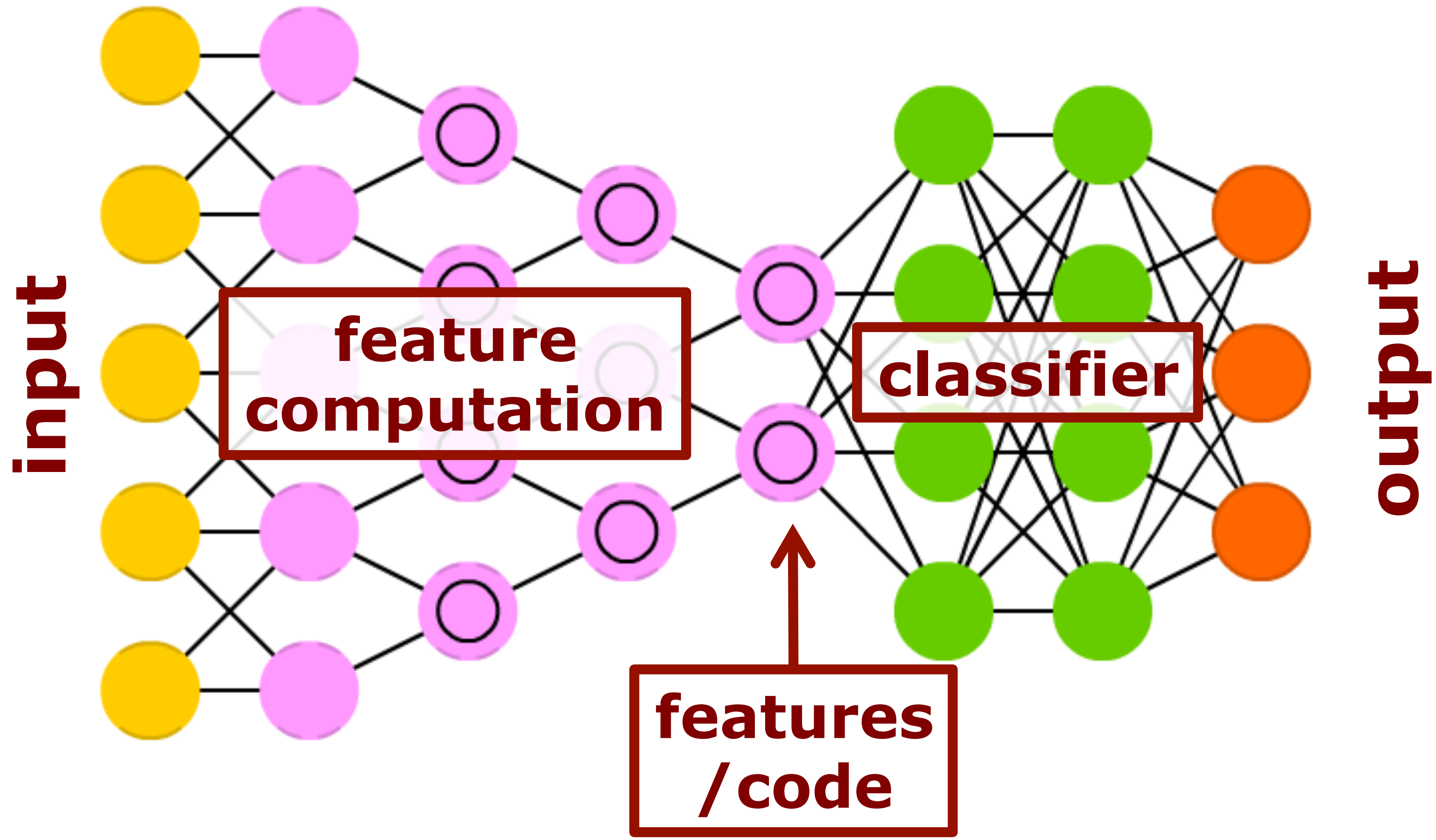
In image-related learning tasks, CNNs play an important role



[Image courtesy: van Veen]

Convolutional Neural Networks

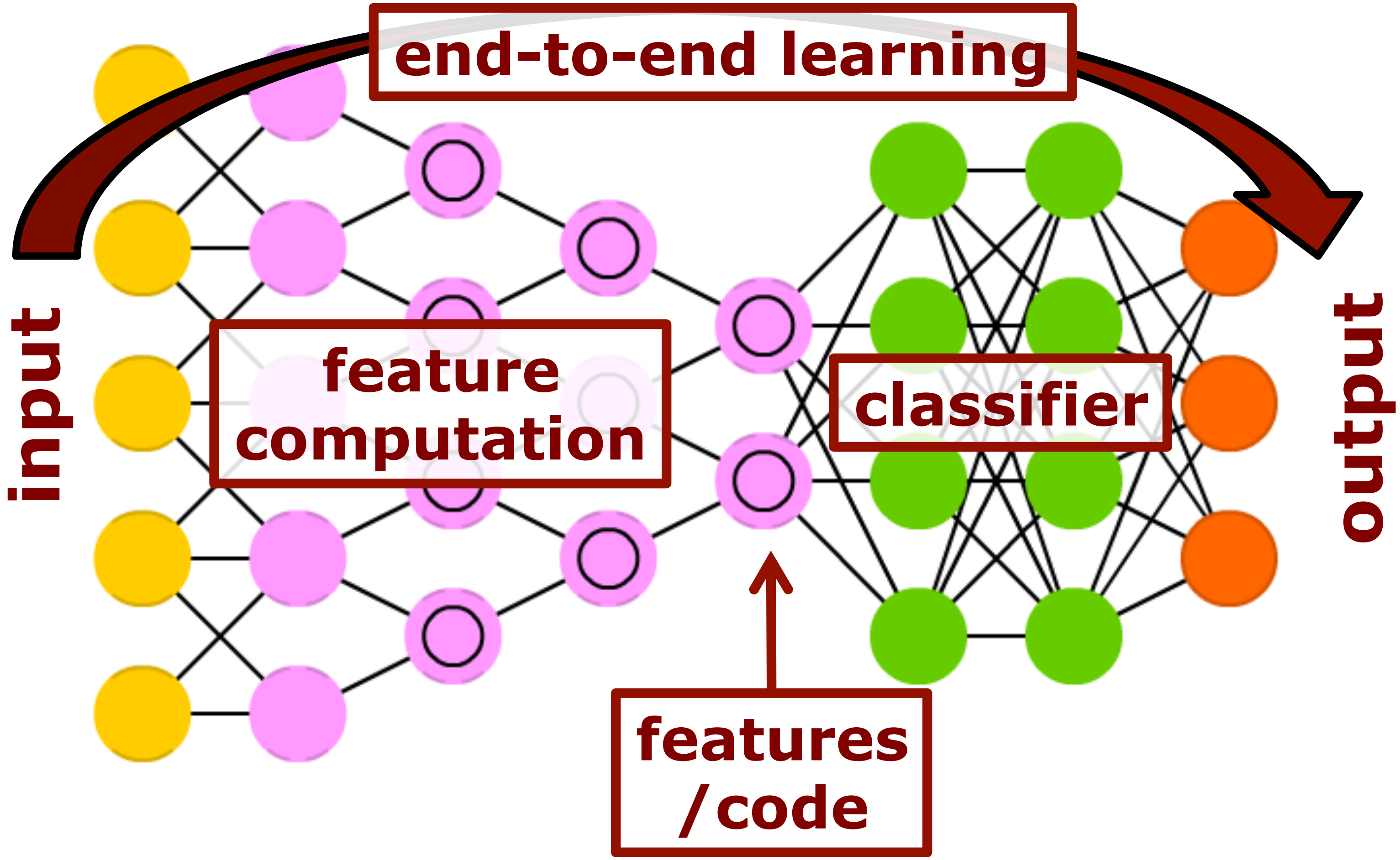
In image-related learning tasks, CNNs play an important role



[Image courtesy: van Veen]

Convolutional Neural Networks

In image-related learning tasks, CNNs play an important role



[Image courtesy: van Veen]

What Is “Deep Learning”?

“Learning neural networks with many hidden layers”

Definition: #hidden layers > 2

Very deep networks today
have up to 150 hidden layers
(still growing…)

Summary – Part 2

- Leaning multi-layer perceptrons
- Parameters are the weights and biases
- Learning = estimate weights & biases
- Minimization of a loss (cost) function
- Gradient descent for parameter optimization
- Backpropagation to compute gradients
- End-to-end: no manual features
- CNN for image processing

Literature & Resources

- Online Book by Michael Nielsen, Chapter 1: http://neuralnetworksanddeeplearning.com/chap1.html
- Nielsen, Chapter 1, Python3 code: https://github.com/MichalDanielDobrzanski/DeepLearningPython
- MNIST database:
- http://yann.lecun.com/exdb/mnist/
- Grant Sanderson, Neural Networks

  https://www.3blue1brown.com/
- Online book by Deisenroth, Faisal, Ong: Mathematics for Machine Learning https://mml-book.github.io/
- Alpaydin, Introduction to Machine Learning
- Standford AI Lectures by Li et al.

Slide Information

- The slides have been created by Cyrill Stachniss as part of the photogrammetry and robotics courses.
- **I tried to acknowledge all people from whom I used images or videos. In case I made a mistake or missed someone, please let me know.**
- Huge thank you to Grant Sanderson (3blue1brown) for his great educational videos that influenced this lecture.
- Thanks to Michael Nielsen for his free online book & code as well as to Fei-Fei Li et al. for the Stanford AI lectures.
- If you are a university lecturer, feel free to use the course material. If you adapt the course material, please make sure that you keep the acknowledgements to others and please acknowledge me as well. To satisfy my own curiosity, please send me email notice if you use my slides.

Cyrill Stachniss, cyrill.stachniss@igg.uni-bonn.de