Deep Learning in *Computer Vision*

What is Computer Vision

- The goal in computer vision is to extract useful information from images
- This includes, but is not limited to:
  - Reconstructing properties such as shape, illumination and color distributions
  - Detecting objects - e.g. faces in images
  - Estimating motion in image sequences
  - Detecting and matching points of special interest between images - e.g. for creating panorama images
  - …

Image classification

Dog

Piano

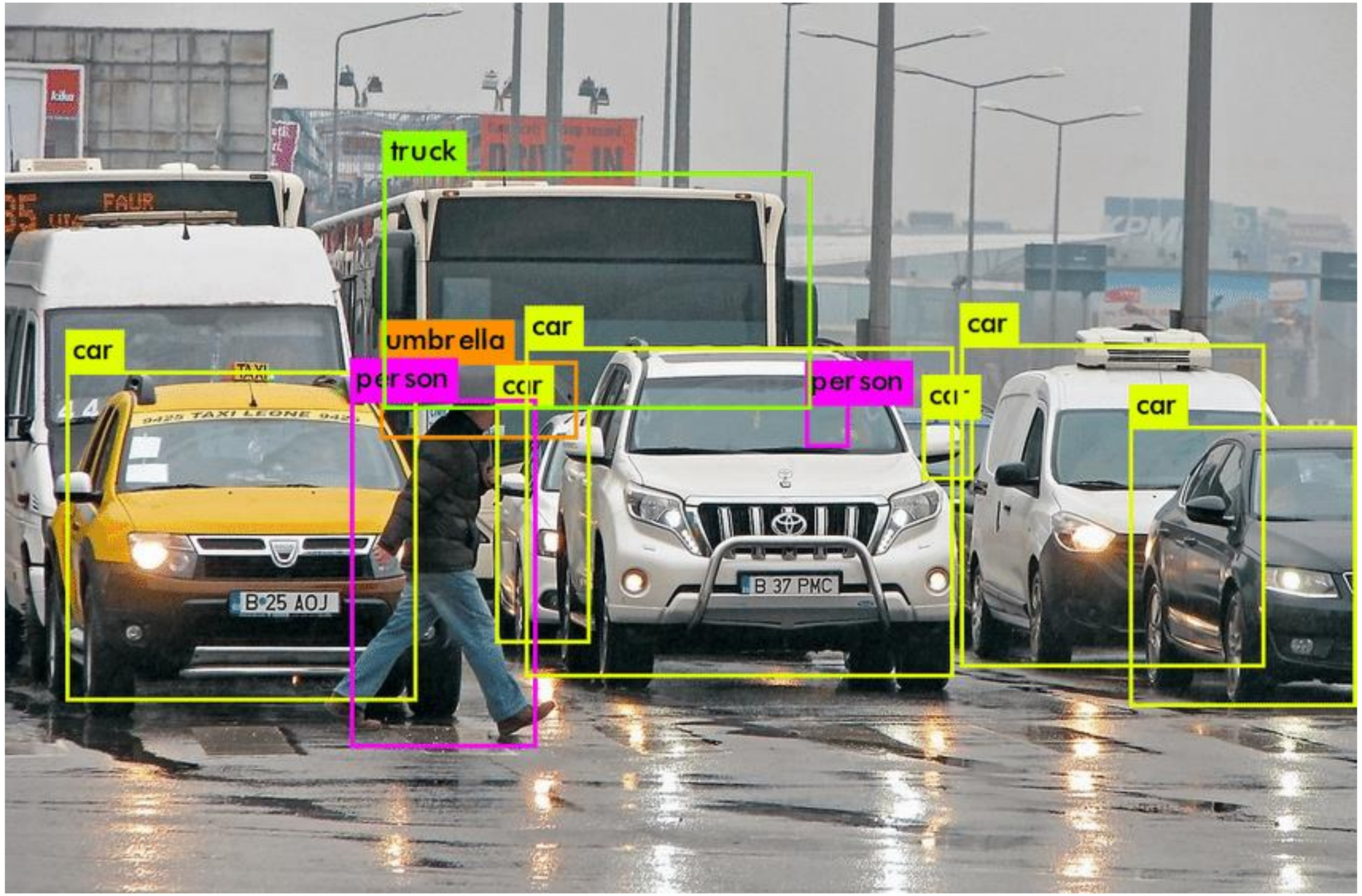
Cake

Computer

Unknown

Image classification

Object Detection



Segmentation

Generative Models


Deep Learning in Computer Vision

Learning

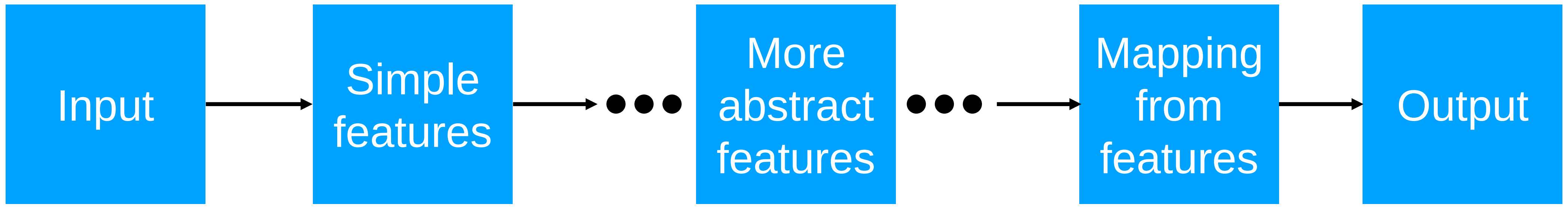
- Supervised learning
  - Learn mapping from input to output given training examples with known output
- Unsupervised learning
  - Learn “something” about the distribution of input examples without having known output
- Reinforcement learning
  - Learn actions based on rewards

Deep Learning

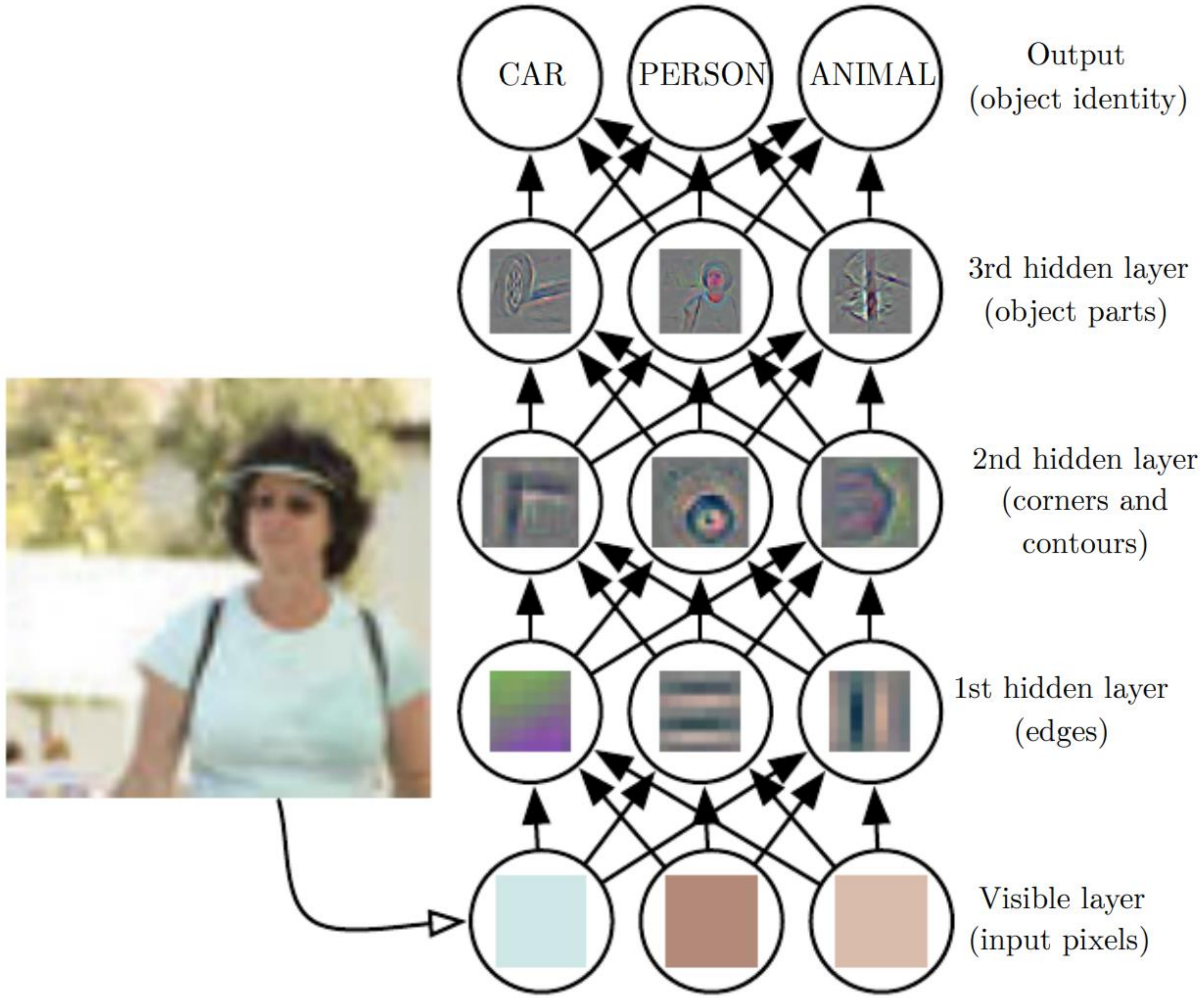
- Deep learning uses deep neural networks
  - Neural networks can approximate any function, if they have enough parameters

Neural Networks

- In deep learning we seek to map a set of input values to output values
- Going directly from input to output is in most cases not possible
- Instead, we learn representations of the inputs from which it is easier to predict the output
- In deep learning we learn increasingly complex representations/features that are expressed in terms of simpler representations/features



Neural Networks



Deep Learning, Goodfellow et al.

Neural networks

- The previous example can be written as:

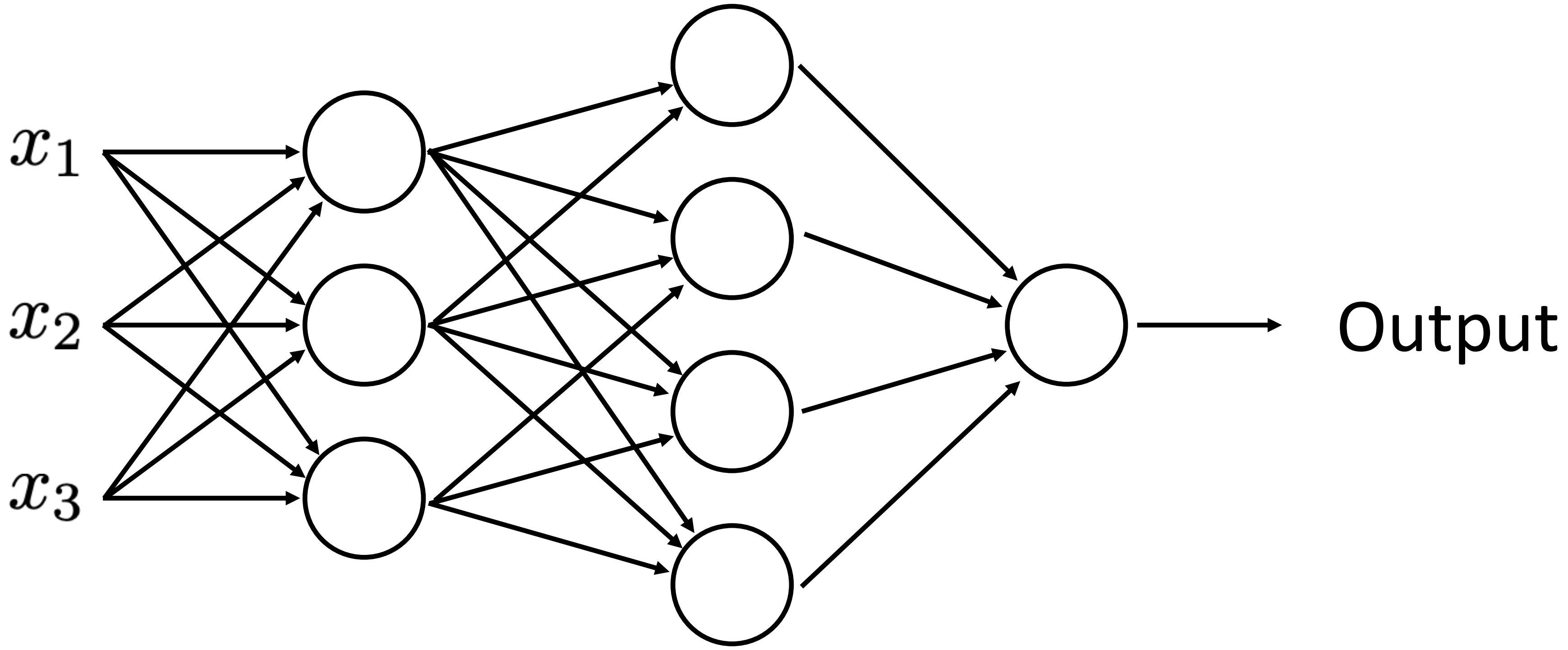
$$f(x) = f_4(f_3(f_2(f_1(x))))$$

Output

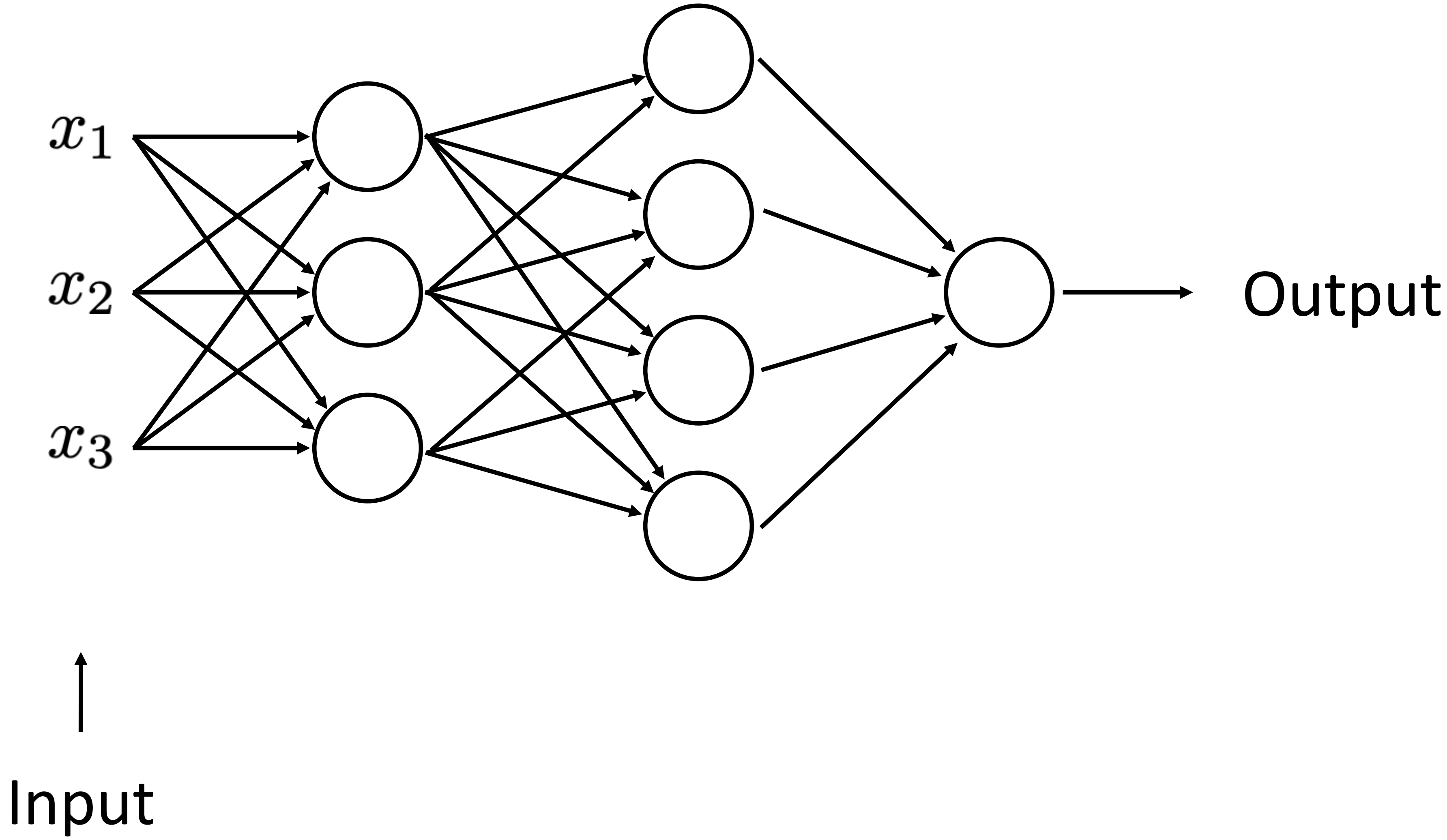
Input

- How do we represent the functions f_1, f_2, f_3, f_4?

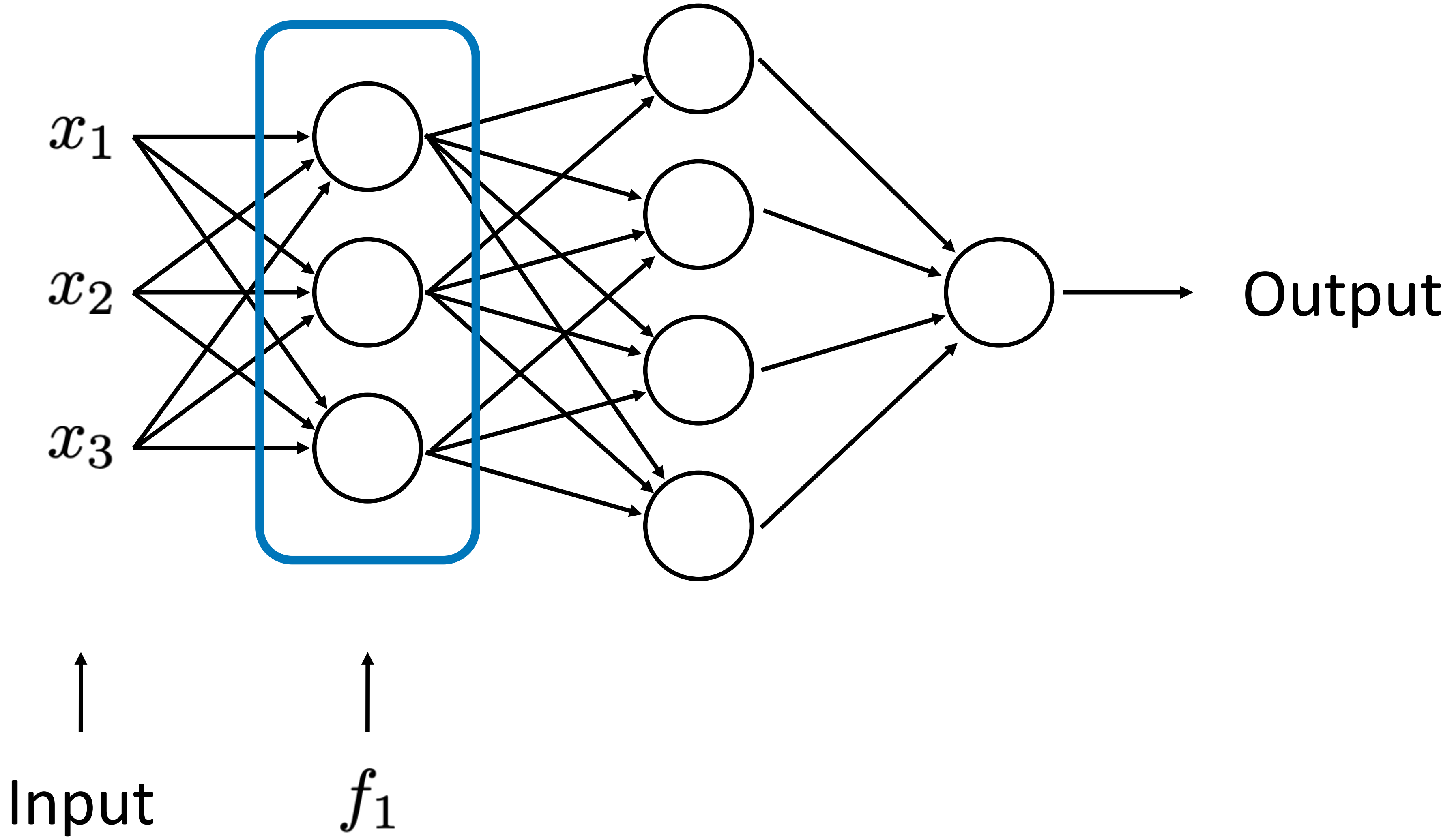
Feed-Forward Neural networks



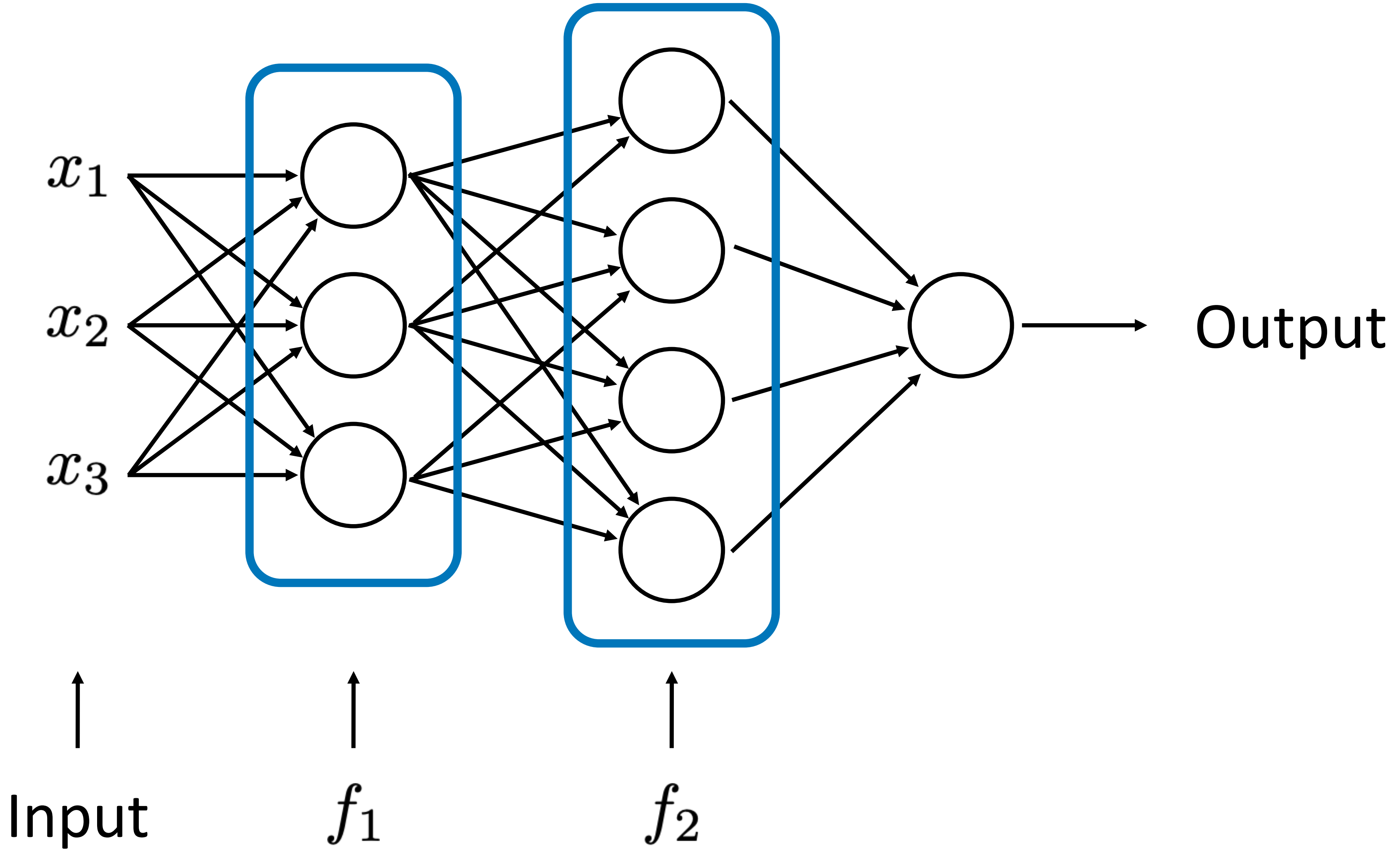
Feed-Forward Neural networks



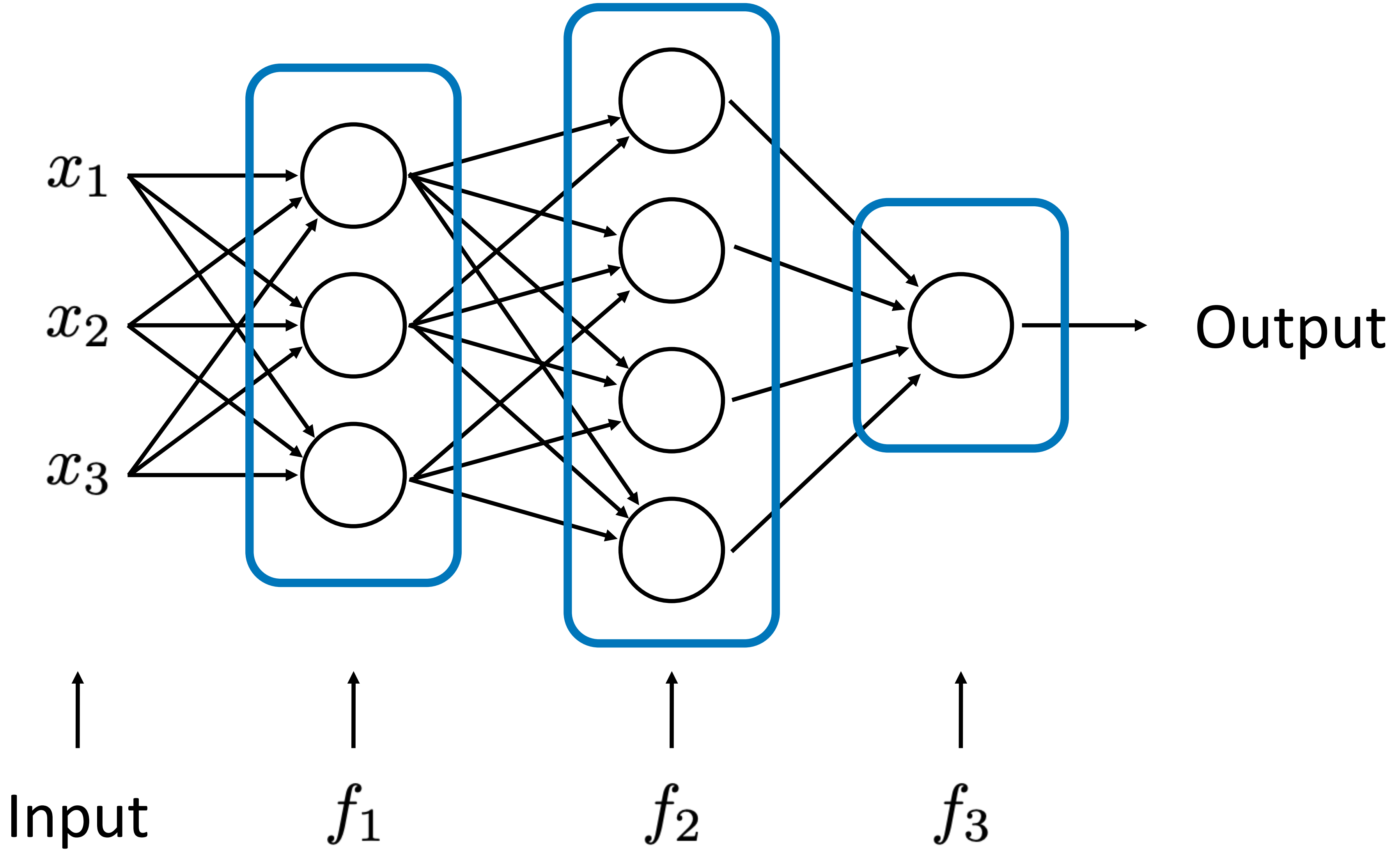
Feed-Forward Neural networks



Feed-Forward Neural networks

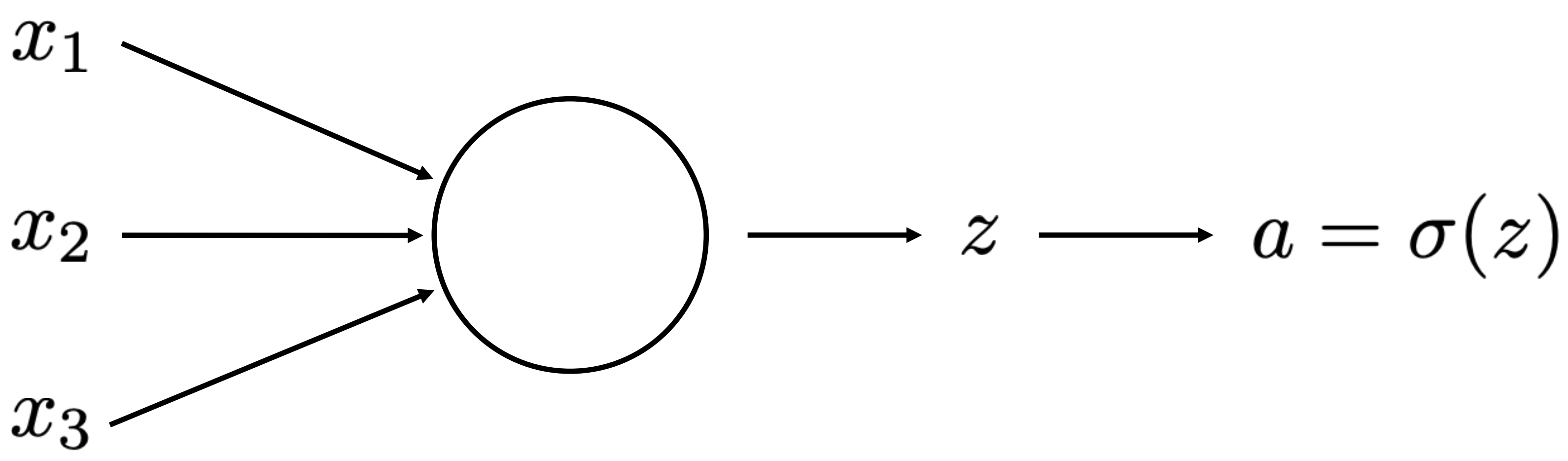


Feed-Forward Neural networks



Neurons - the building block

- Originally inspired by neurons in the brain

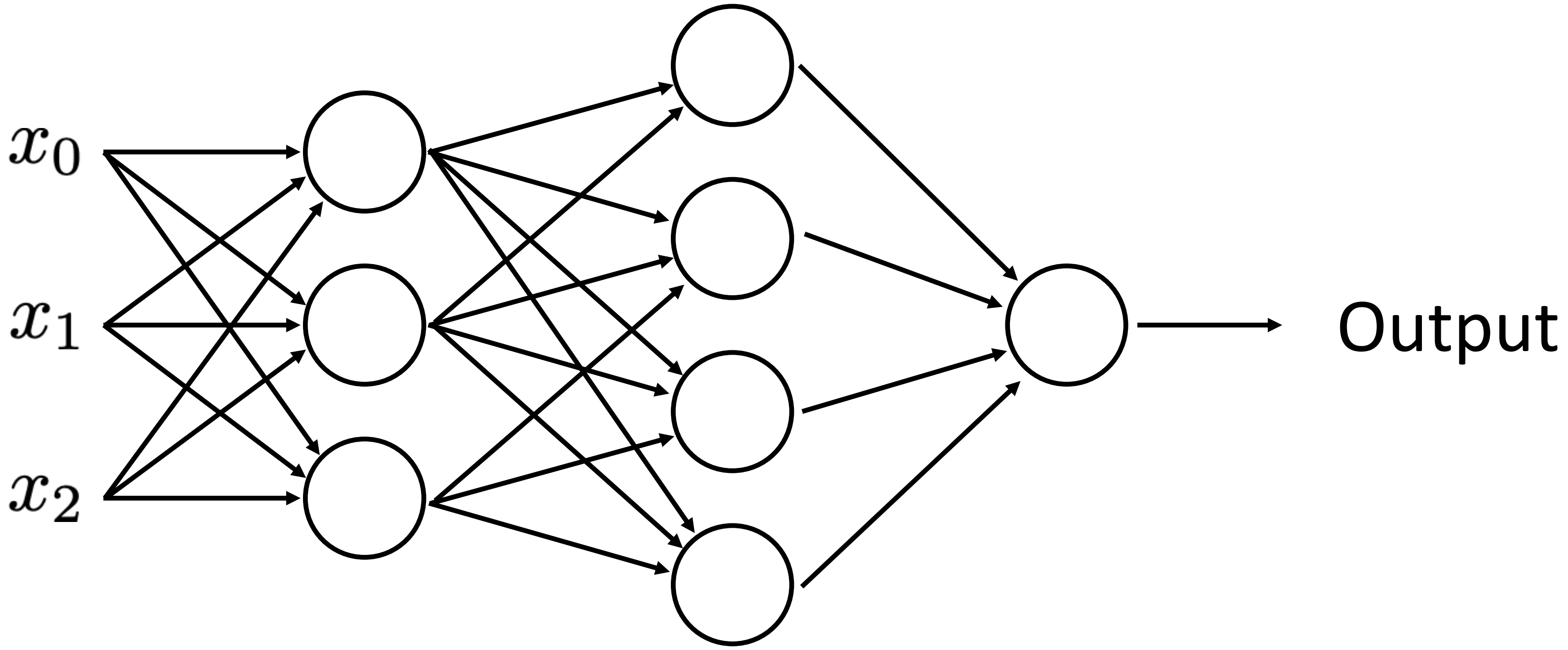


$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b = \mathbf{wx} + b$$

$a = \sigma(z)$ is the output of the neuron

$\sigma(\cdot)$ is a non-linear activation function

Feed-forward computation



$$\mathbf{a}^0 = \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

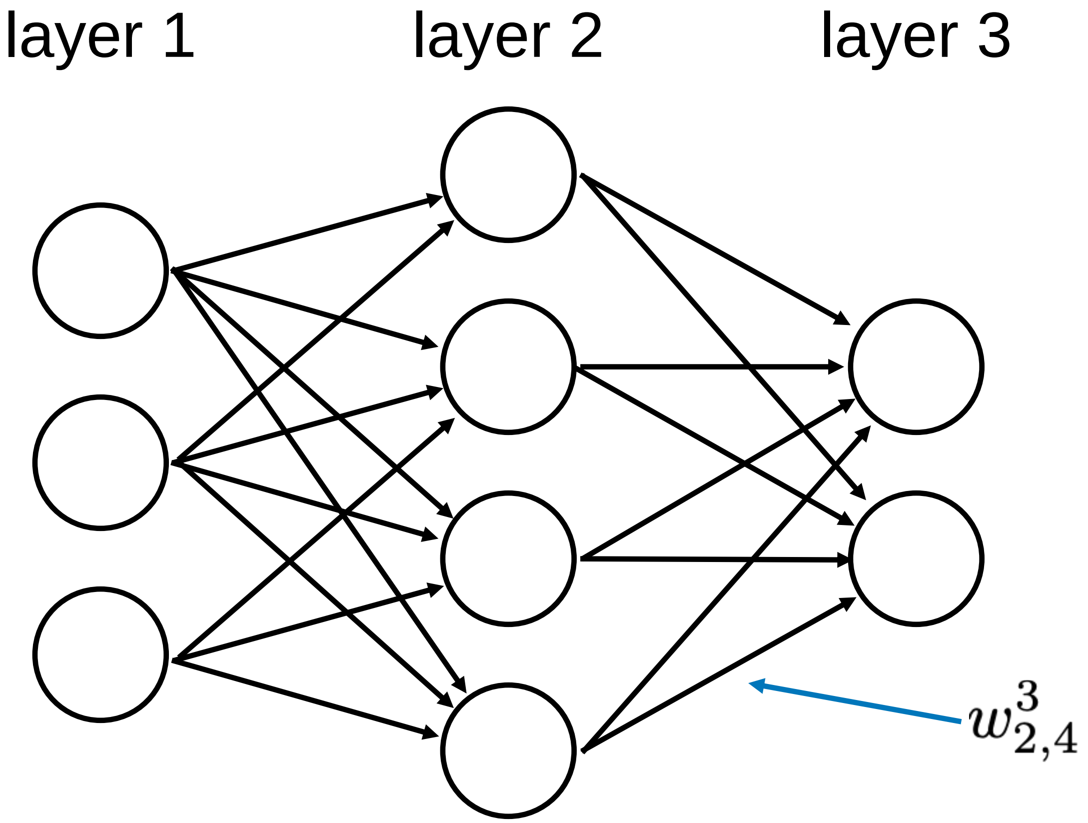
$$\mathbf{a}^1 = \sigma(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1)$$

$$= \sigma\left(\begin{bmatrix} w^1_{0,0} & w^1_{0,1} & w^1_{0,2} \\ w^1_{1,0} & w^1_{1,1} & w^1_{1,2} \\ w^1_{2,0} & w^1_{2,1} & w^1_{2,2} \end{bmatrix} \begin{bmatrix} a^0_0 \\ a^0_2 \\ a^0_2 \end{bmatrix} + \begin{bmatrix} b^1_0 \\ b^1_1 \\ b^1_2 \end{bmatrix}\right)$$

$$\mathbf{a}^2 = \sigma(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2)$$

$$\ldots$$

Feed-forward computation



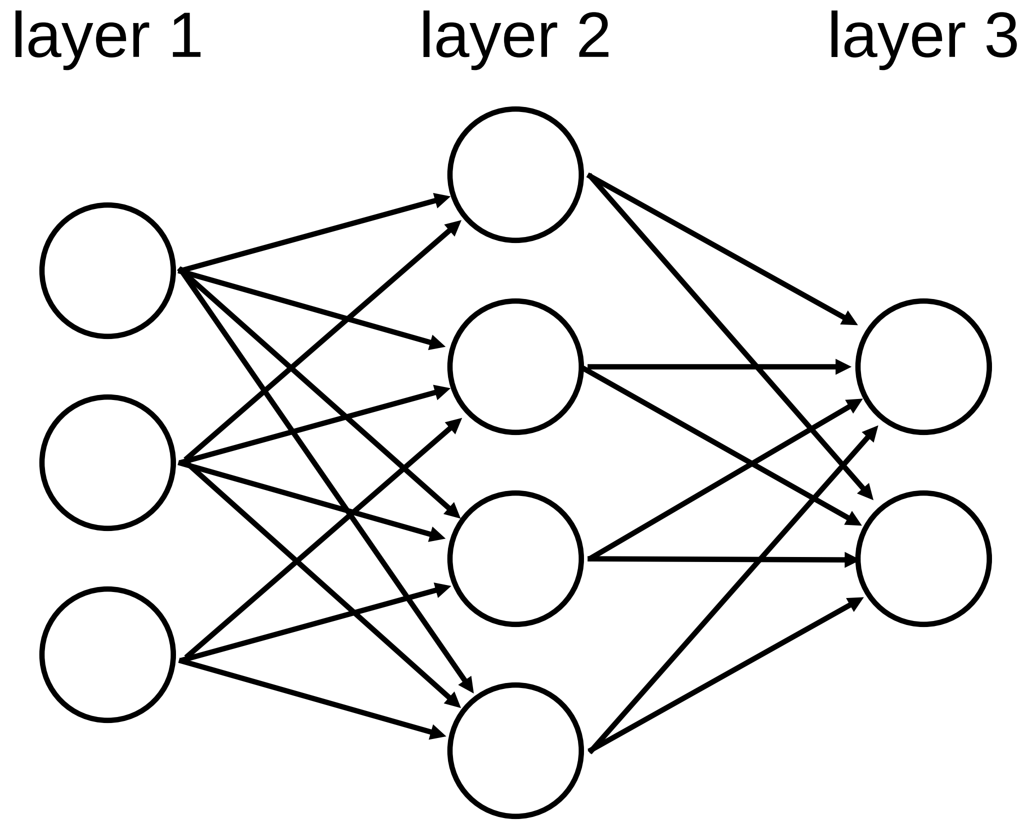
$$\mathbf{a}^\ell = \sigma(\mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell)$$
$$= \sigma\left(\sum_k w^\ell_{j,k} a^{\ell-1}_k + b^\ell_j\right)$$

$w^\ell_{j,k}$ is the weight from the k^{th} neuron in the $(l\text{-}1)^{th}$ layer to the j^{th} neuron in the l^{th} layer

b^ℓ_j is the bias of the j^{th} neuron in the l^{th} layer

a^ℓ_j is the activation of the j^{th} neuron in the l^{th} layer

Feed-forward computation



- We need a non-linear activation function after each layer
  - Otherwise, the entire network is a linear model (i.e., only one layer)

How do we obtain the parameters?

- Right now, the model is just a lot of math, but how do we choose the weights and biases in the model, so it works well on our data?
- We need a function (loss) that measures how well our network is doing for a given set of parameters W and b
- How can we use this to learn the parameters?

How do we obtain the parameters?

- Random search?
  - Try many different random weights and biases and choose the best one
- Random local search?
  - Generate random weights and biases close to our current and choose the best one
- Gradient Descent
  - Use the gradient of our loss function, which gives the direction of maximum descent at any point *x*

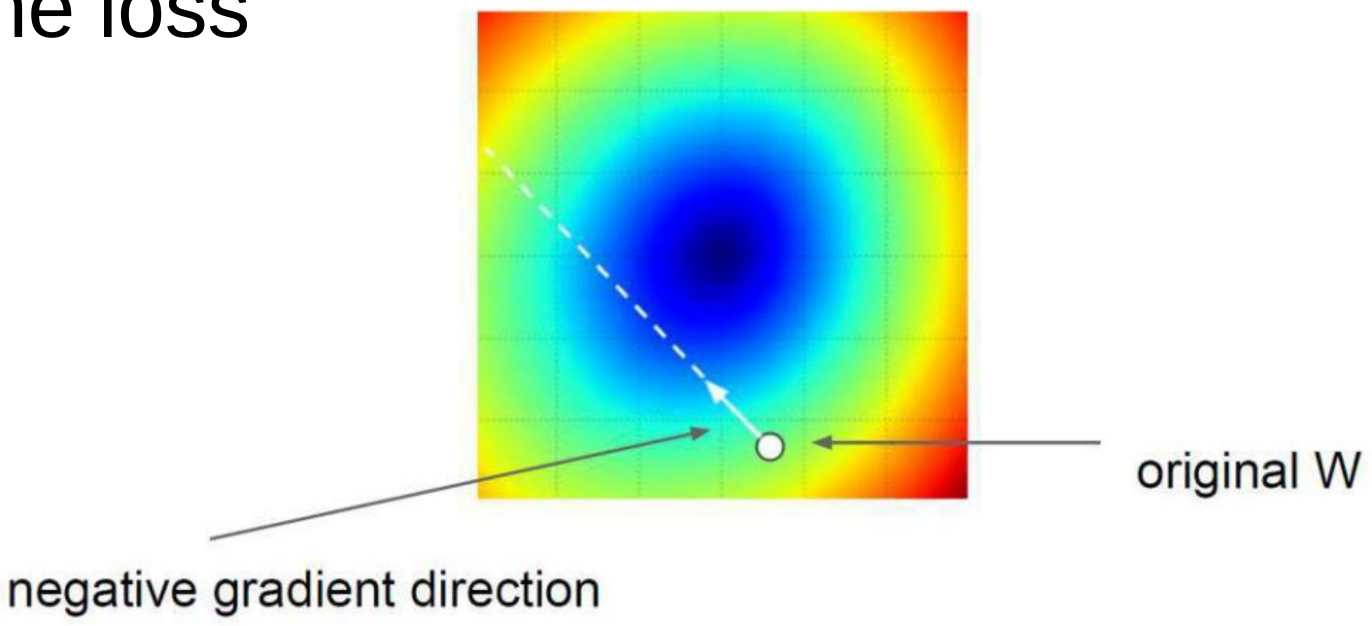
How do we obtain the parameters?

- Random search?
  - Try many different random weights and biases and choose the best one
- Random local search?
  - Generate random weights and biases close to our current and choose the best one
- **Gradient Descent**
  - **Use the gradient of our loss function, which gives the direction of maximum descent at any point *x***

Gradient descent

- **Gradient** descent
  - The gradient of a parameter, describes how the loss will change if we change this parameter

- Gradient **descent**
  - We change the value of the parameter slightly in the direction that should decrease the loss



Gradient descent

- If $\mathcal{L}$ is the loss function, we wish to minimize with respect to parameters *p*
  - The gradient $\nabla_p \mathcal{L}$ gives us the direction of maximum ascent of the loss function
  - $-\nabla_p \mathcal{L}$ is the direction of maximum descent
- Gradient descent algorithm:
  - Compute the gradient
  - Take a step in the negative gradient direction

$$p \to p' = p - \alpha \nabla_p \mathcal{L}$$

  - Here α is called the learning rate and determines the step size
  - Repeat until convergence

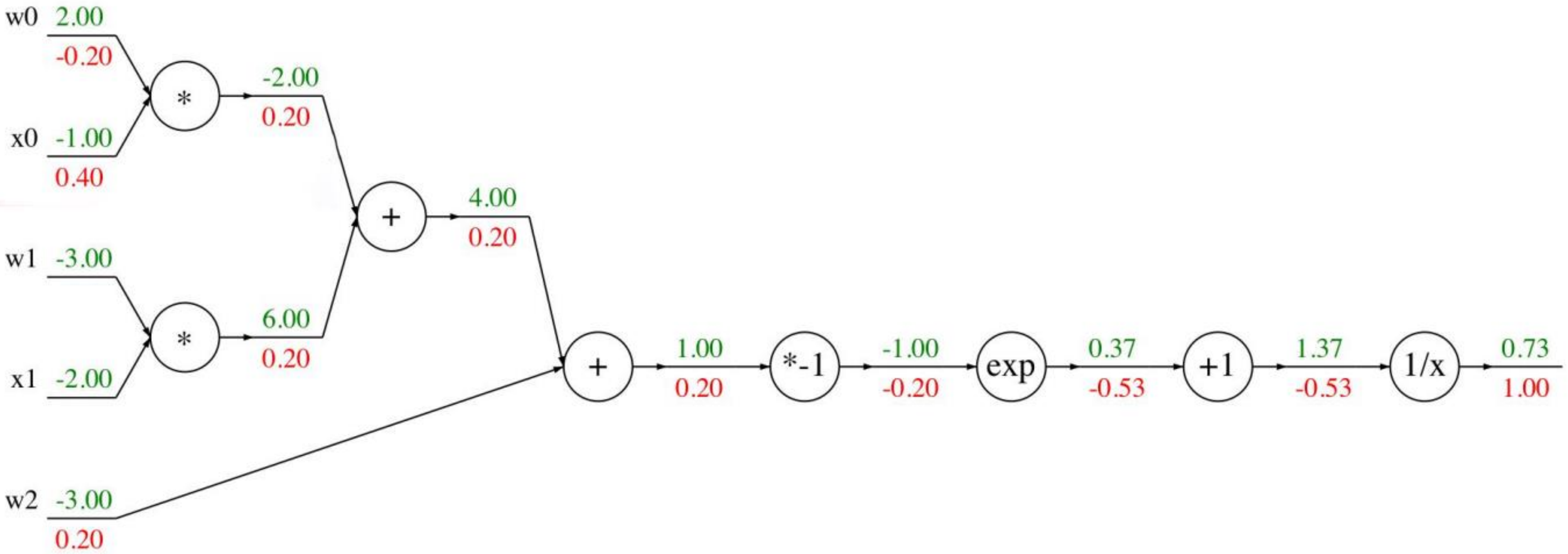
Backpropagation

- Which parameters do we want to find the partial derivates of $\mathcal{L}$ with respect to?
  - The weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$
- How?
  - We propagate the error back through the network
    - Recursive approximation of the chain rule

Short break

Backpropagation example

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



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

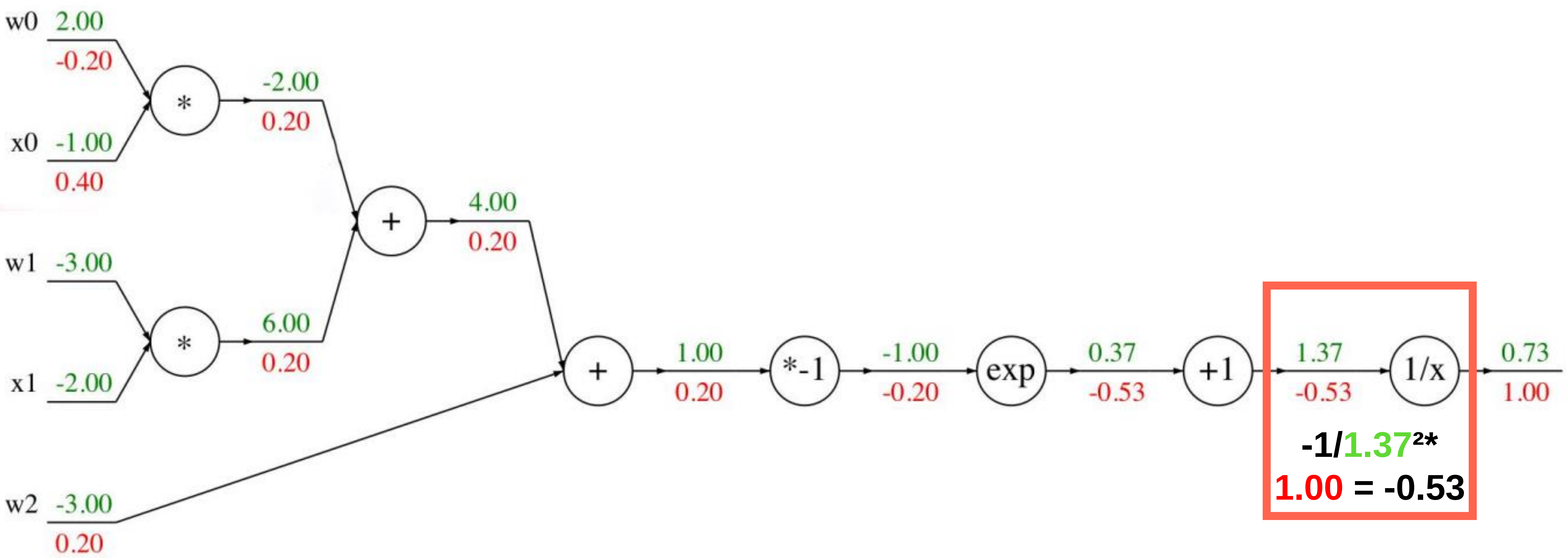
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Backpropagation example

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



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

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

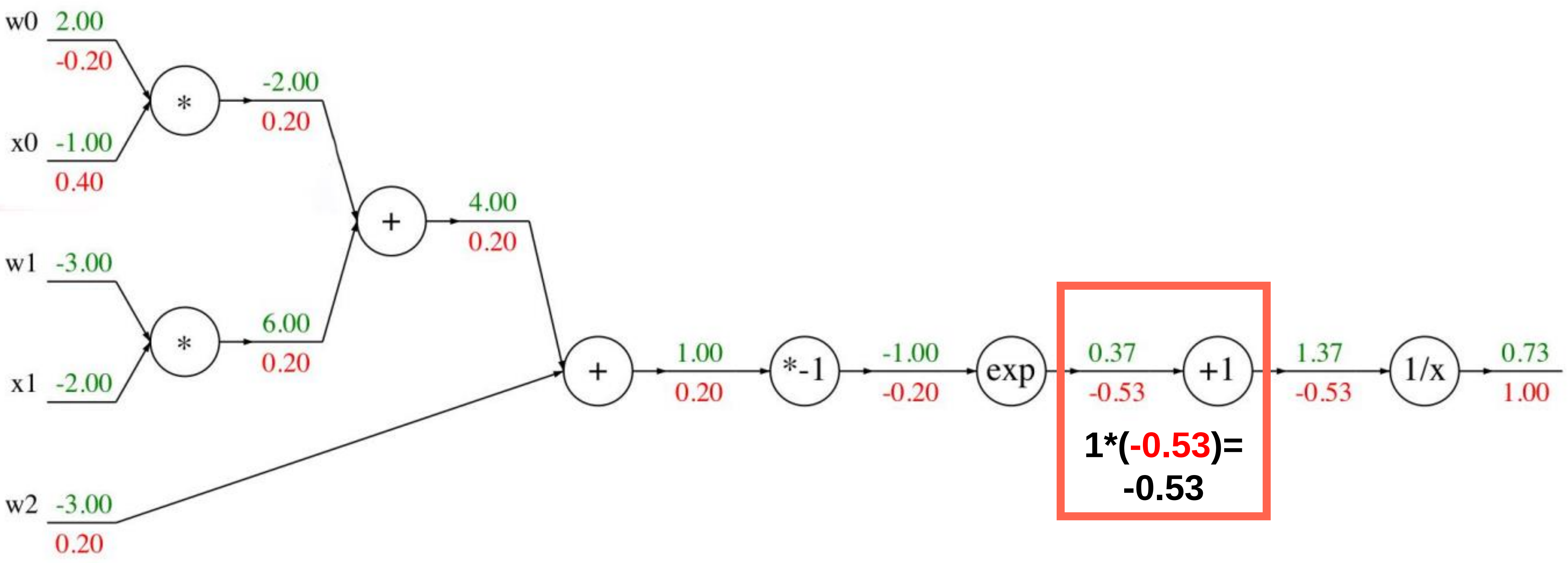
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Slide credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, CS231n: Convolutional Neural Networks for Visual Recognition.

Backpropagation example

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



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

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

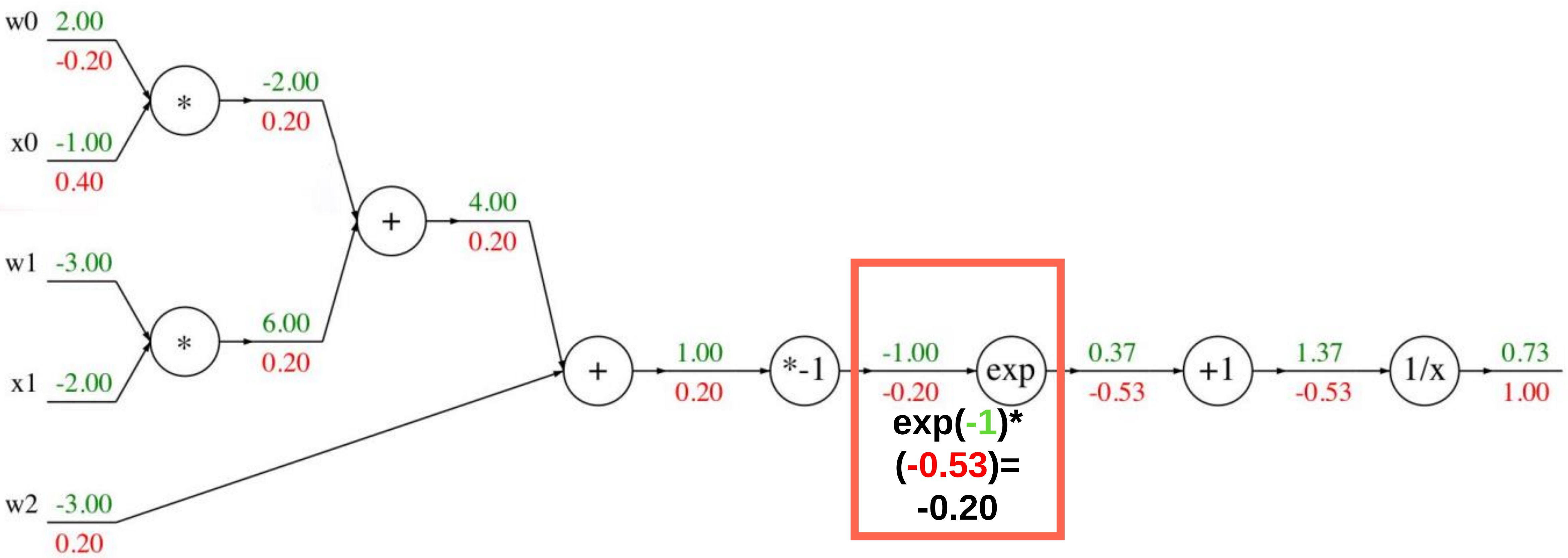
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Slide credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, CS231n: Convolutional Neural Networks for Visual Recognition.

Backpropagation example

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



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

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

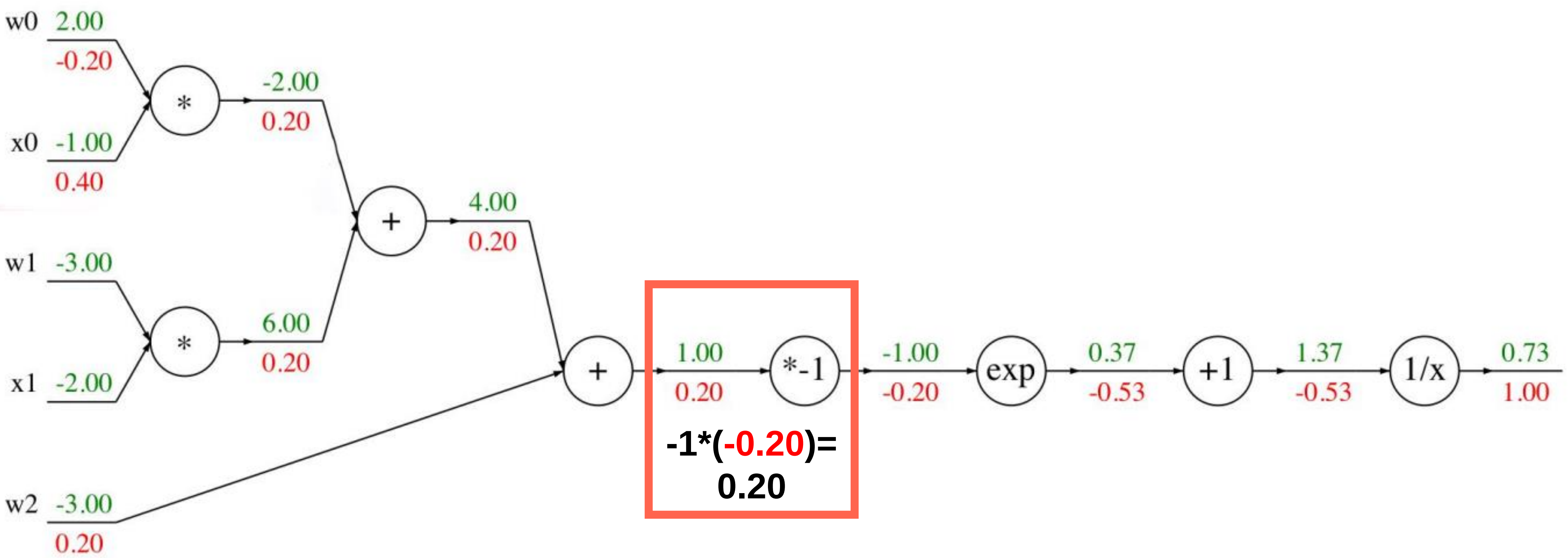
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Slide credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, CS231n: Convolutional Neural Networks for Visual Recognition.

Backpropagation example

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



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

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

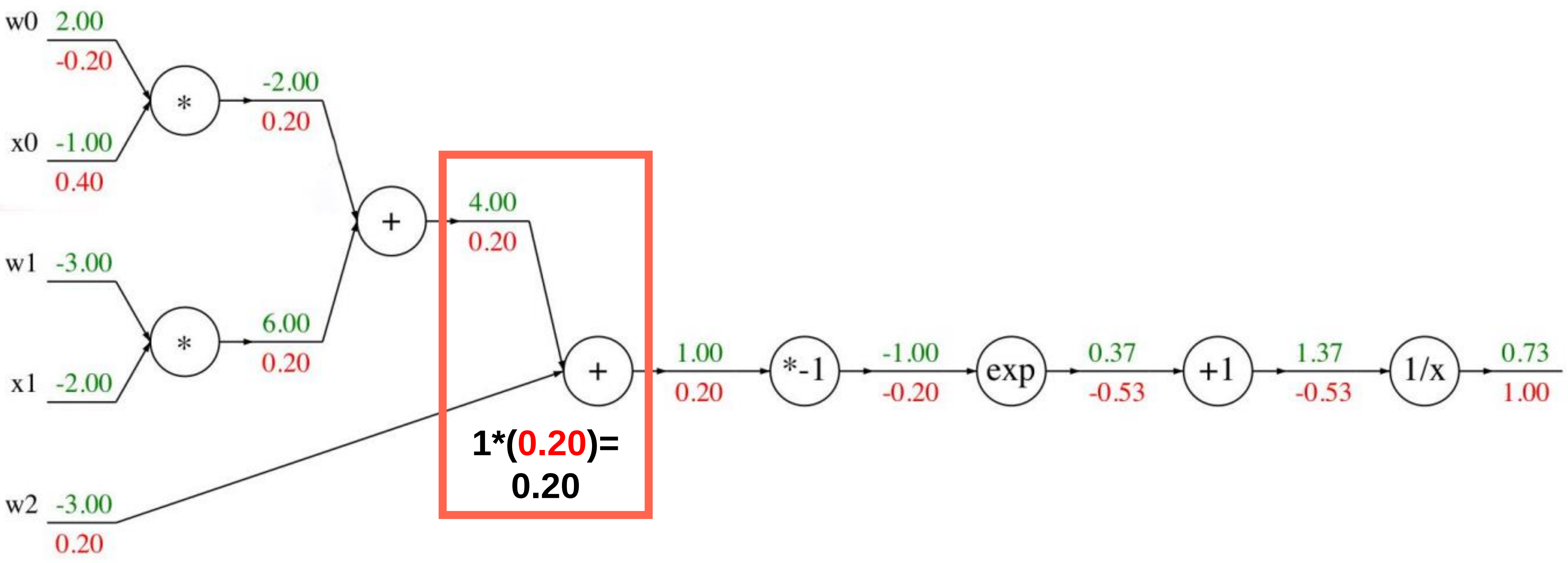
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Slide credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, CS231n: Convolutional Neural Networks for Visual Recognition.

Backpropagation example

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

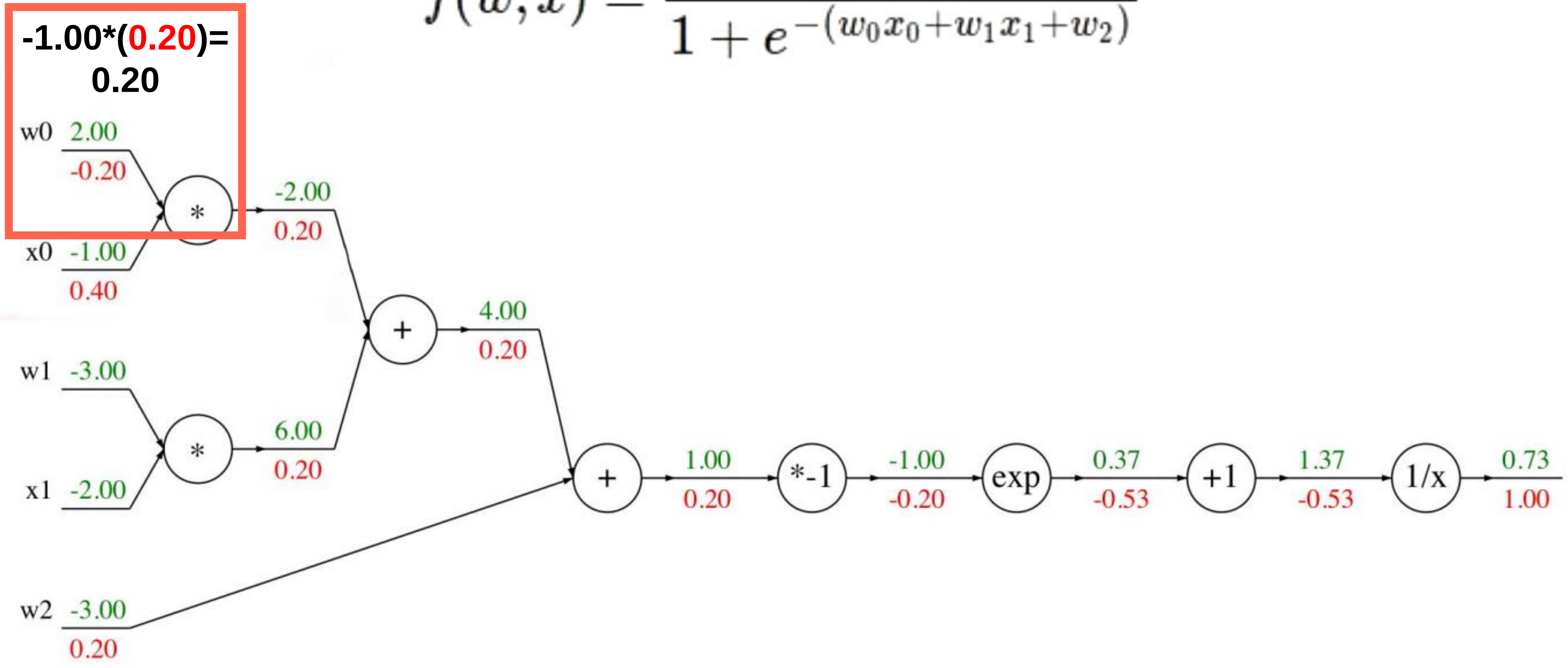


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Backpropagation example

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



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

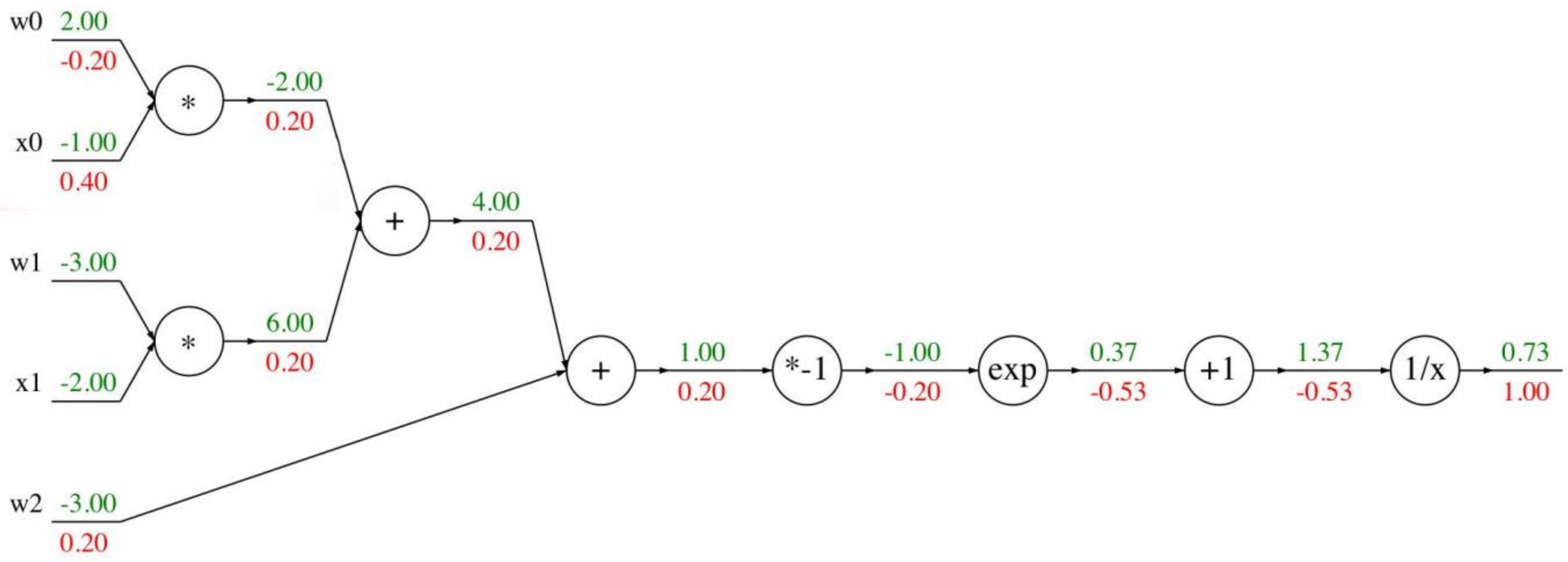
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Backpropagation example

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



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

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

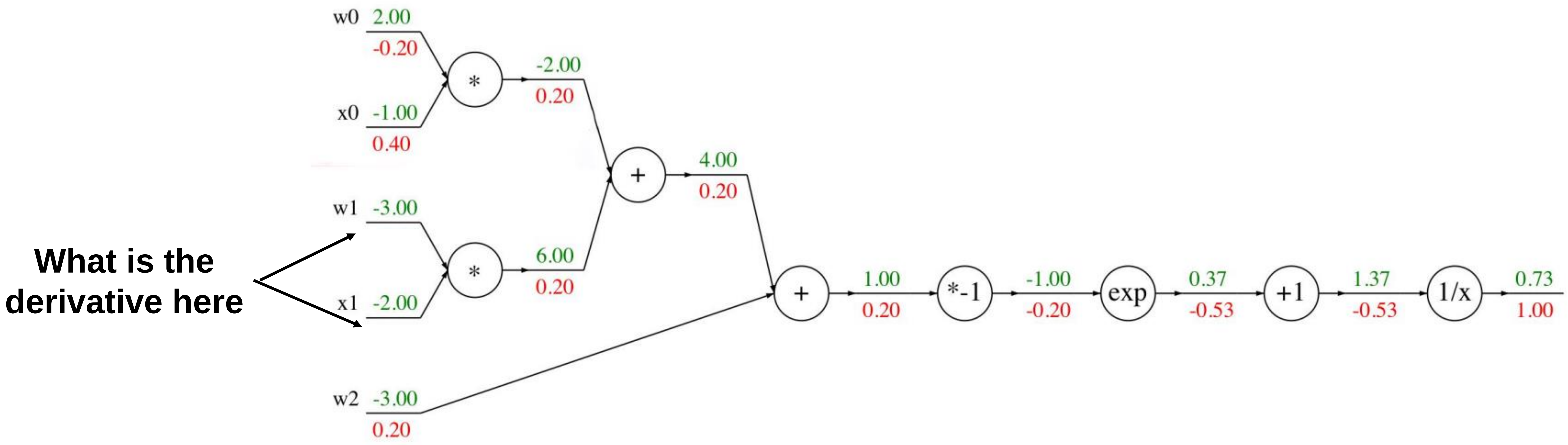
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Slide credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, CS231n: Convolutional Neural Networks for Visual Recognition.

Backpropagation example

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



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

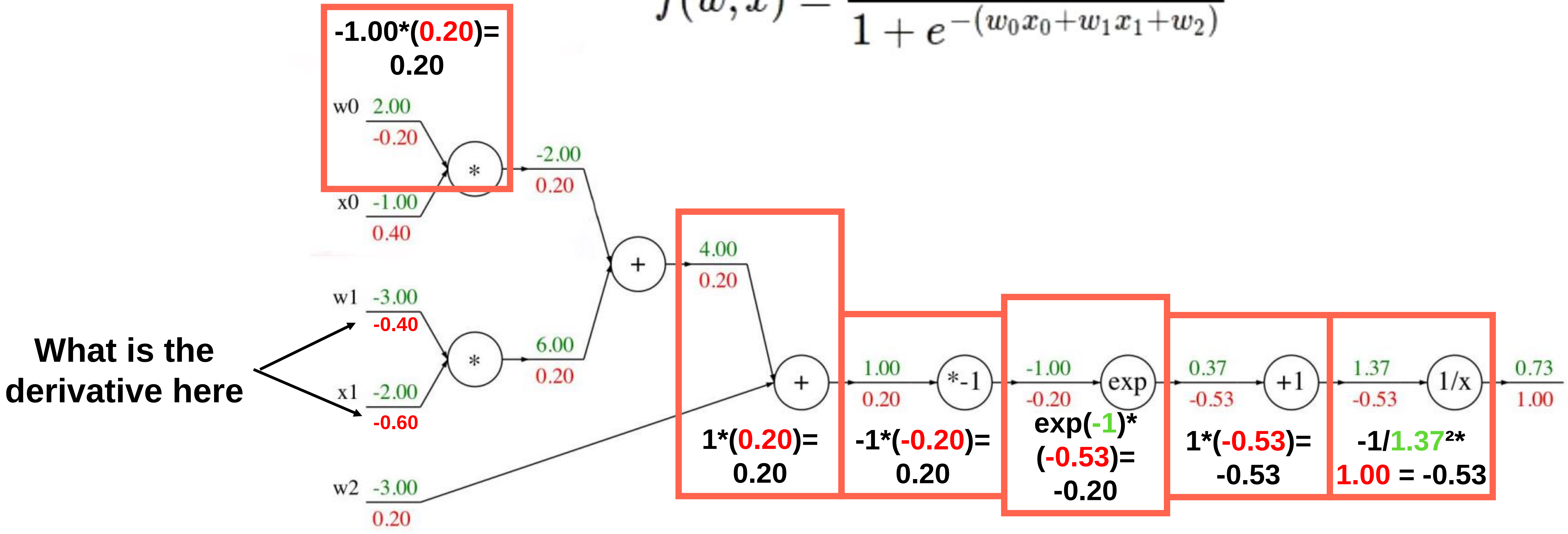
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Backpropagation example

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



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

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

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

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

Slide credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, CS231n: Convolutional Neural Networks for Visual Recognition.

The equations of backpropagation

- Our goal is to find the partial derivatives of the loss function $\mathcal{L}$ with respect to any weight $w^{\ell}_{j,k}$ or bias b^{ℓ}_{j}
- The forward pass is defined by $z^{\ell+1}_{k} = \sum_{j} w^{\ell+1}_{k,j} \sigma(z^{\ell}_{j}) + b^{\ell+1}_{k}$
- By the chain rule we have that

$$\frac{\partial \mathcal{L}}{\partial w^{\ell}_{j,k}} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}} \frac{\partial z^{\ell}_{j}}{\partial w^{\ell}_{j,k}} = a^{l-1}_{k} \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}}$$

$$\frac{\partial \mathcal{L}}{\partial b^{\ell}_{j}} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}} \frac{\partial z^{\ell}_{j}}{\partial b^{\ell}_{j}} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}}$$

- We thus need a way to calculate $\frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}}$

The equations of back propagation

- For the last layer in our network this is easily done (*L* is the last layer)

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\sigma'(z_j^L)$$

- The first term after the last equals sign depends on both the choice of loss function and choice of activation function

The equations of back propagation

- Recall: $$z_k^{\ell+1} = \sum_j w_{k,j}^{\ell+1} \sigma(z_j^\ell) + b_k^{\ell+1}$$

- For the rest of the layers in the network we have ($\odot$ is elementwise product)

$$\frac{\partial \mathcal{L}}{\partial z_j^\ell} = \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell}$$

$$= \sum_k w_{k,j}^{\ell+1} \sigma'(z_j^\ell) \frac{\partial \mathcal{L}}{\partial z_k^{\ell+1}}$$

$$\frac{\partial \mathcal{L}}{\partial z^\ell} = ((\mathbf{W}^{\ell+1})^T \frac{\partial \mathcal{L}}{\partial z^{\ell+1}}) \odot \sigma'(z^\ell)$$

Activation functions

- The two most commonly used activation functions are:

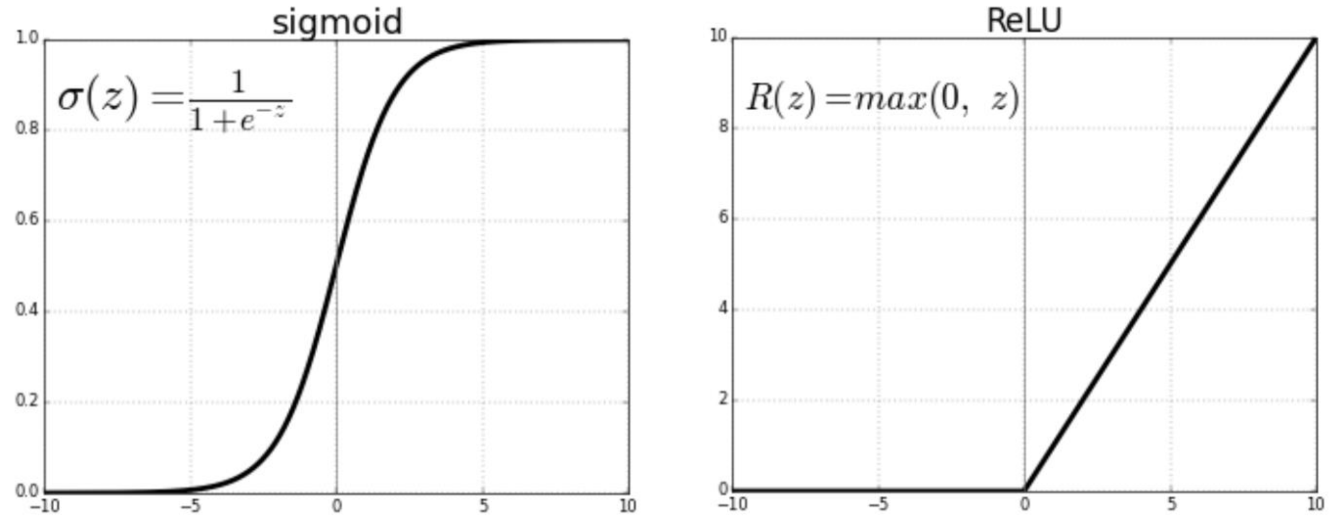
- Sigmoid

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

  Closer to how the brain works
  Vanishing gradient problem

- Rectified Linear Unit: $\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$

  Works well in most cases
  Not differentiable at zero
    Not a problem in practice



Derivatives of activation functions

- Sigmoid

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

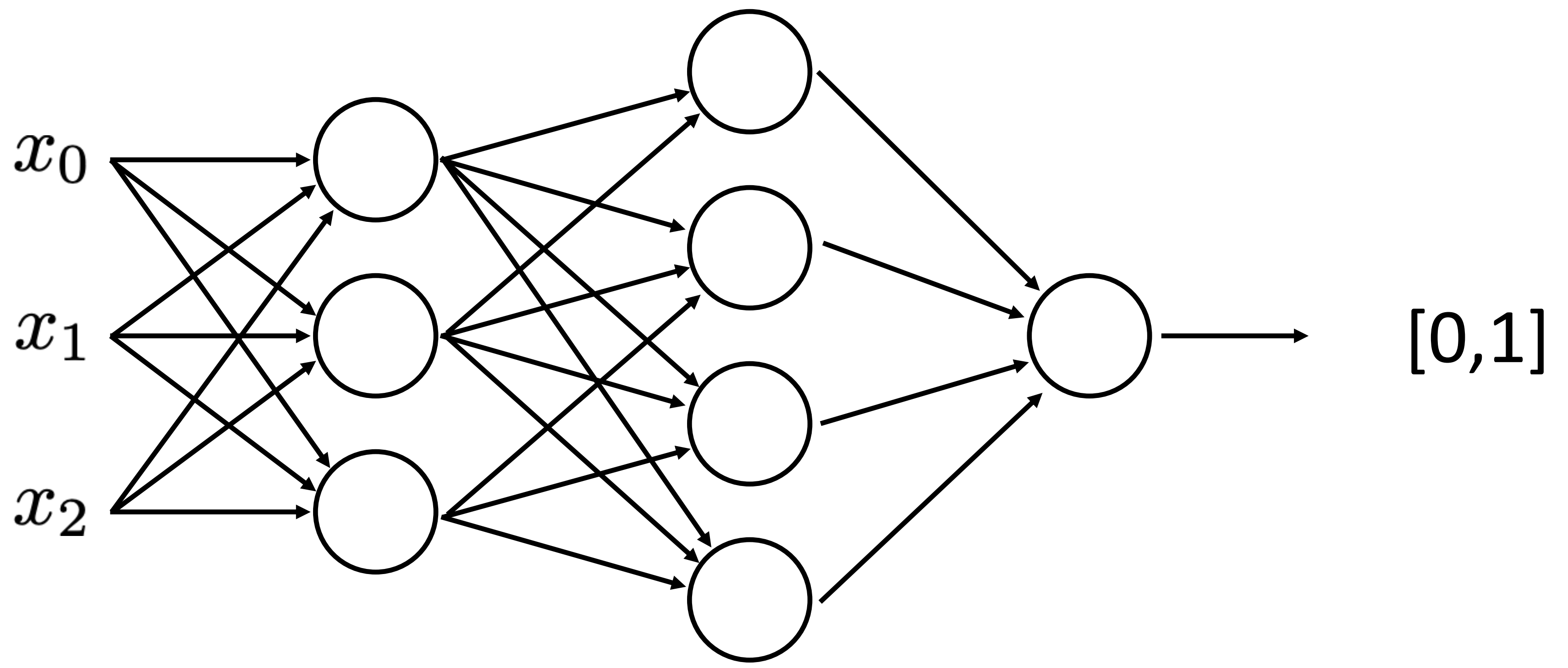
- Rectified Linear Unit:

$$\sigma(x) = \mathrm{ReLU}(x) = \max(0, x) \qquad \sigma'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Loss functions

- The choice of loss functions depends on the task
  - For regression: L2 (squared) or L1 (absolute)
  - For two-class classification: Binary cross-entropy
  - For multi-class classification: Cross-entropy
  - …

Two-class classification



- The output from the neuron in the last layer is mapped to the range [0,1] by using a sigmoid activation function on last neuron

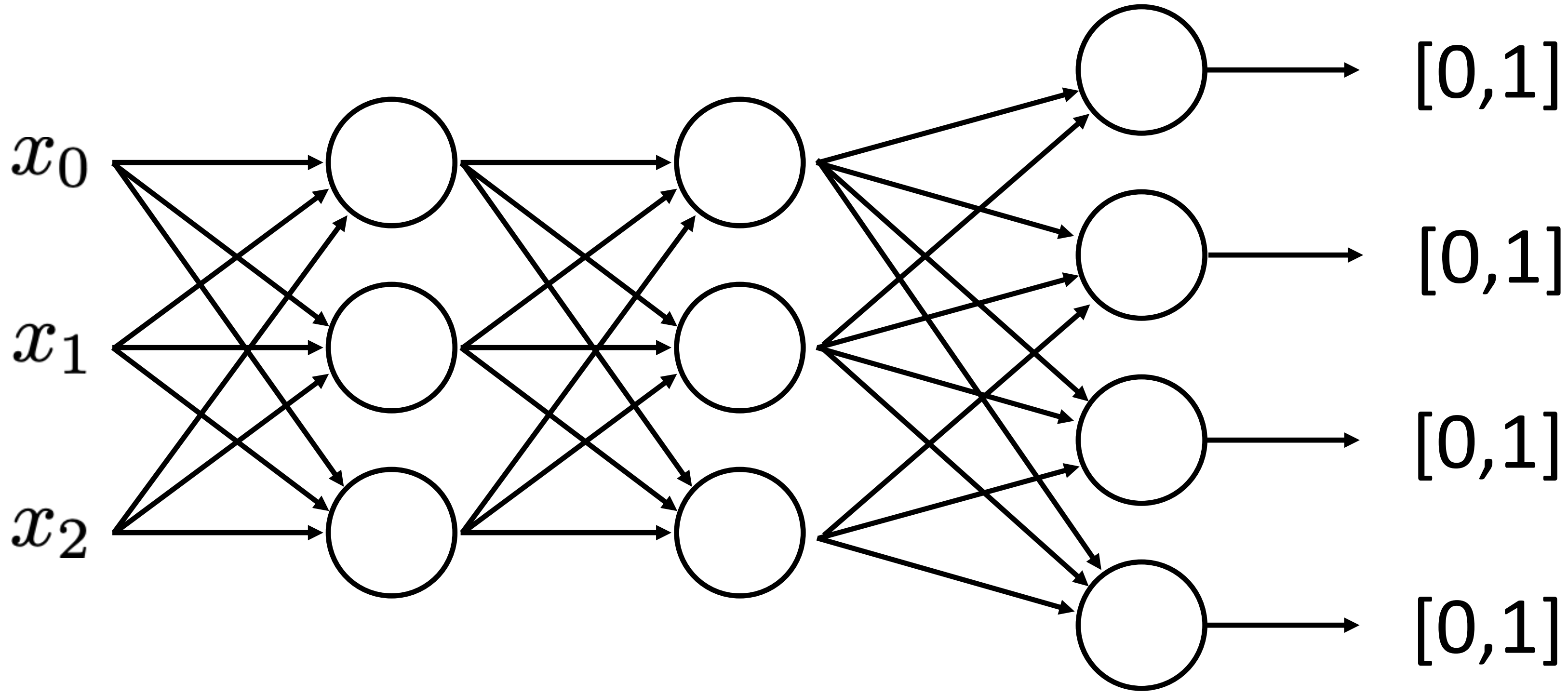
Binary cross-entropy

- If $\hat{y} \in [0, 1]$ is the output from a neural network and $y \in \{0, 1\}$ is the true value for an input *x* the binary cross-entropy is given by

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

$$\frac{\partial \mathcal{L}}{\partial a^L} = \frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

Multi-class classification



- The output from the neurons in the last layer is mapped to the range [0,1] such that they sum to 1 by using a softmax activation function in the last layer

Softmax activation function

- The softmax activation function maps the outputs from all neurons in layer to the range [0,1] such that they sum to 1

$$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$\sum_i x_i = 1$$

- Softmax is shift invariant (if you add a constant to all x, the probabilites are the same)

Cross-entropy

- The cross-entropy is given by

$$\mathcal{L}(\hat{y}, y) = -\sum_i y_i \log \hat{y}_i = -\log \hat{y}_I, \quad I = \arg_i(y_i = 1)$$

- i.e., the negative log likelihood

- Remember that to start the backpropagation we need to compute

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L)$$

Updating the parameters

- The parameters can now be updated by taking a small step in the negative gradient direction

$$w^{\ell}_{j,k} \rightarrow \bar{w^{\ell}_{j,k}} = w^{\ell}_{j,k} - \alpha \frac{\partial L}{\partial w^{\ell}_{j,k}}$$

$$b^{\ell}_{j} \rightarrow \bar{b^{\ell}_{j}} = b^{\ell}_{j} - \alpha \frac{\partial L}{\partial b^{\ell}_{j}}$$

Stochastic gradient descent

- So far, we only discussed how to train a network based on a single example
  - This is called *stochastic gradient descent*
- We can train over multiple training examples by simply averaging the loss and gradients over the examples
- If we use all our training examples, we call it *batch gradient descent*
- If we use random subsets of our examples, it is called *mini-batch gradient descent*
  - *Note: this is often called stochastic gradient descent (which is wrong, but widespread)*
- In practice we always use mini-batch gradient descent
  - The size of the minibatches is a hyperparameter
  - Typically chosen as large as RAM allows

You have learned

- What neural networks are
- How to find the parameters for a neural network from data
  - Loss functions
  - Backpropagation

Forming groups

If you do not have a group of 4 students, please go to the designated area and try to find one. Please try to align:

- Your level of ambition
- Your expected work hours

It is fine if you have different background/expertise. This can even be an advantage!