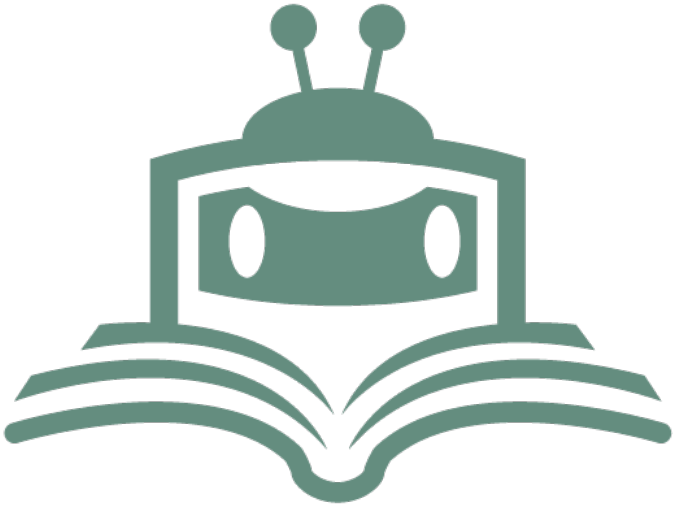


MATHEMATICS FOR MACHINE LEARNING

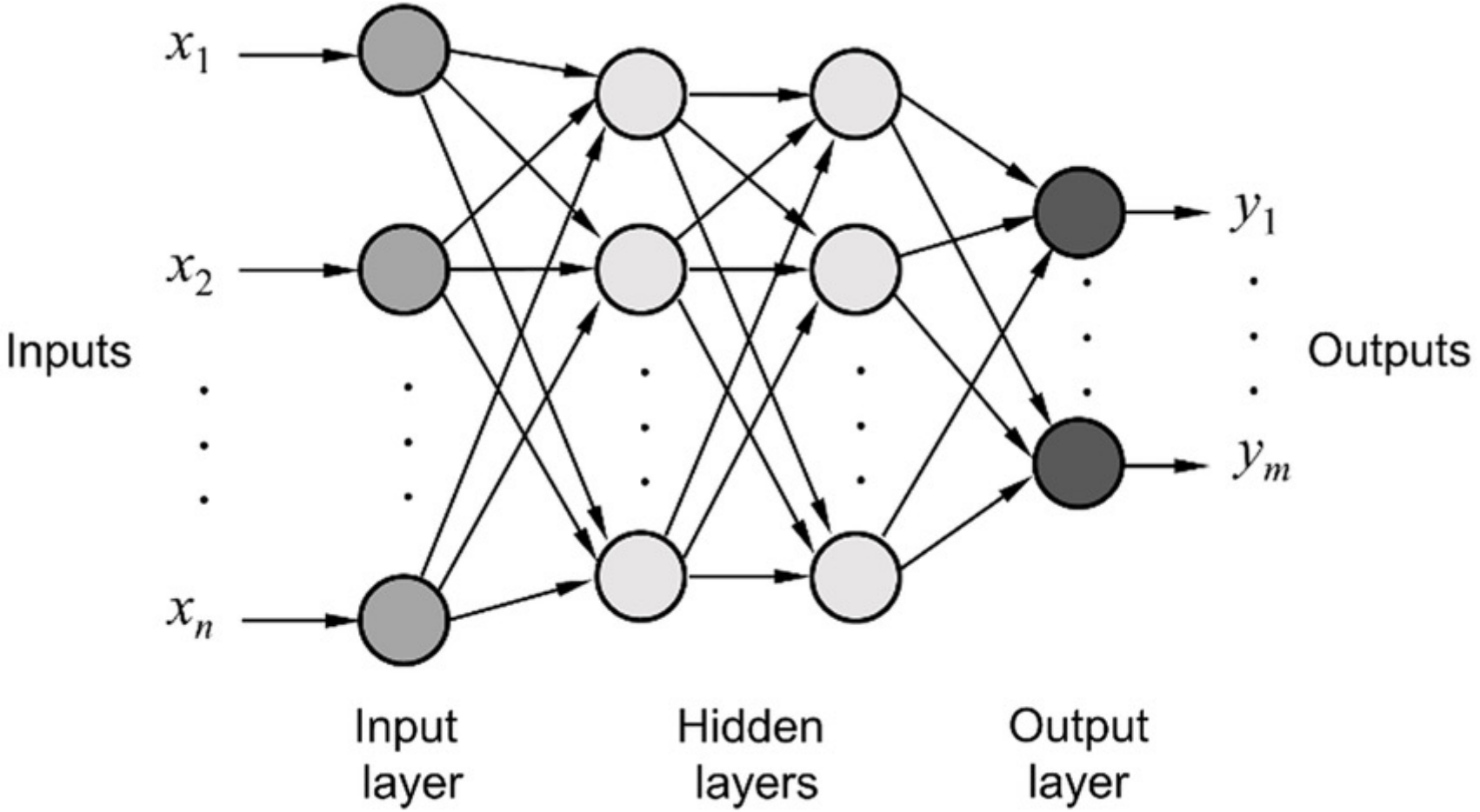
Nagoya University, Fall 2022

Lecture 12

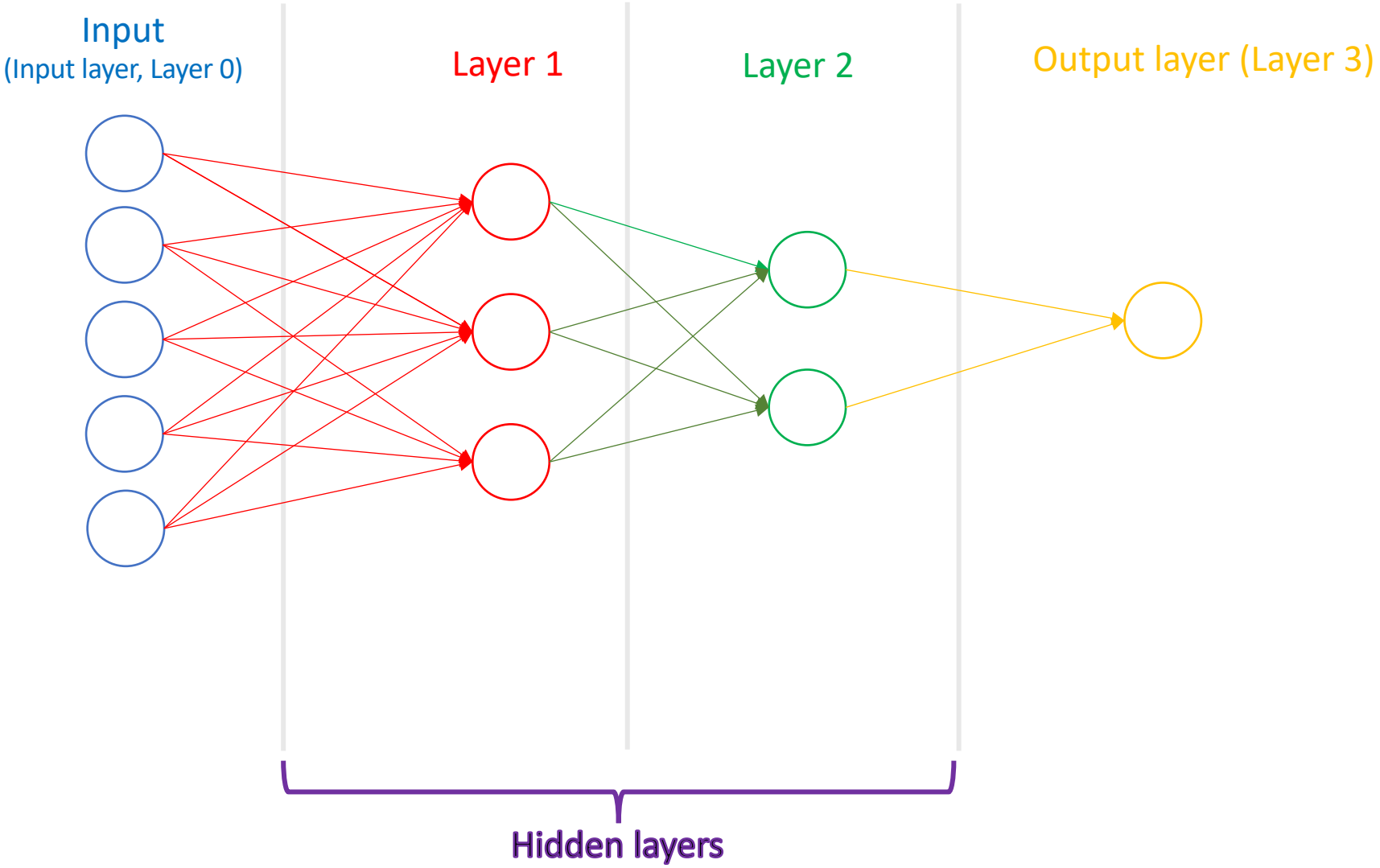
Neural Networks II

https://www.henrikbachmann.com/mml2022.html

Neural Networks



Neural Network with 3 layers



Activation function

Definition 6.1. *(i) An* **activation function** *is a (usually non-linear) function* $\sigma : \mathbb{R}^n \to \mathbb{R}^n$. *Often activation functions are given as functions* $\sigma : \mathbb{R} \to \mathbb{R}$, *which are then extended to a function* $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ *by setting*

$$\sigma \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} := \begin{pmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_n) \end{pmatrix} .$$

(ii) The **rectified linear unit (ReLU)** *is the activation function defined for* $x \in \mathbb{R}$ *by*

$$\mathrm{ReLU}(x) = \max\{0, x\}.$$

(iii) The **sigmoid** *function is the activation function defined for* $x \in \mathbb{R}$ *by*

$$S(x) = \frac{1}{1 - e^{-x}} .$$

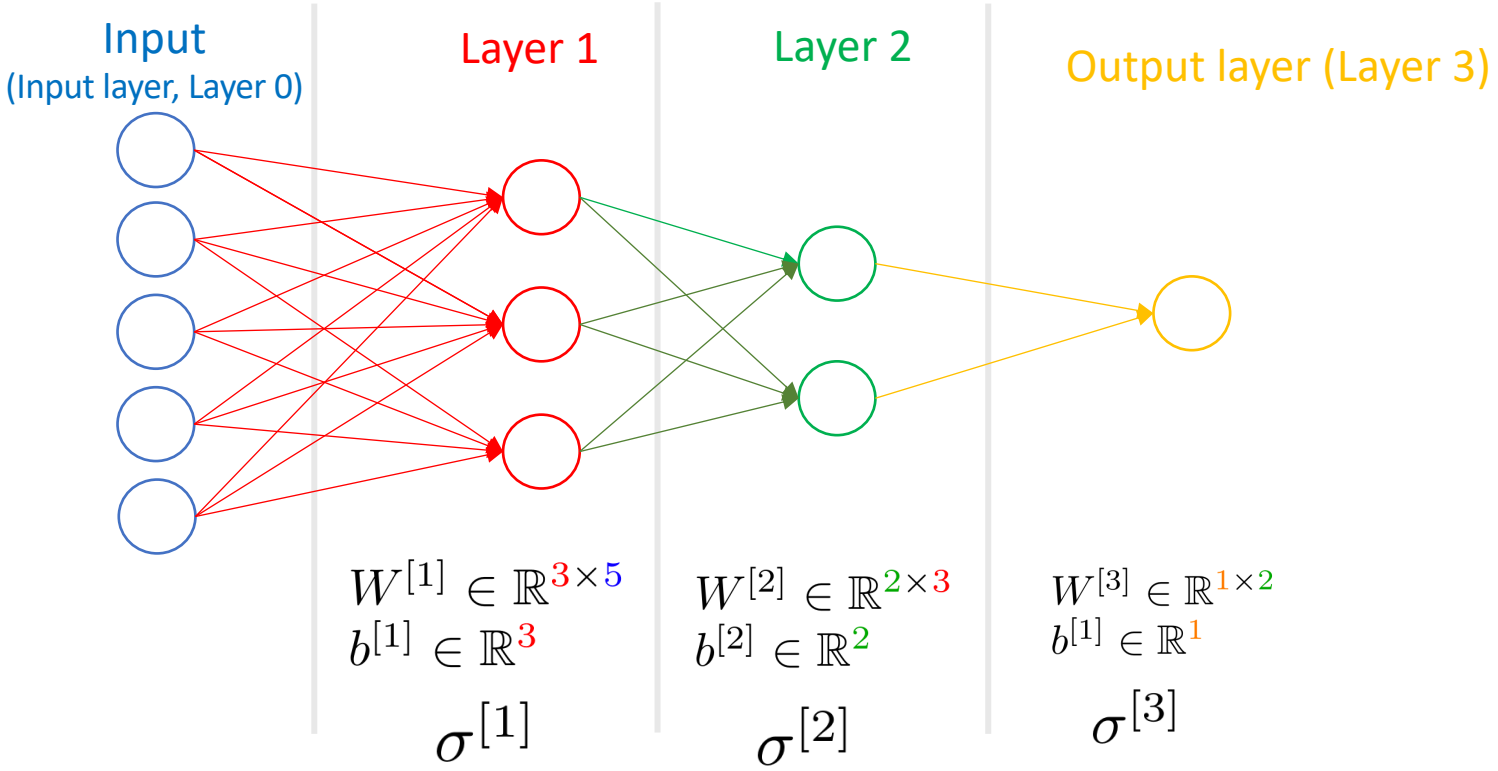
Layer

Definition 6.2. *A* **layer** $L = (W, b, \sigma)$ *of input size* $n \geq 1$ *and (output)* **size** $m \geq 1$ *consists of*

i) *a* **weight matrix** $W \in \mathbb{R}^{m \times n}$,

ii) *a* **bias vector** $b \in \mathbb{R}^m$,

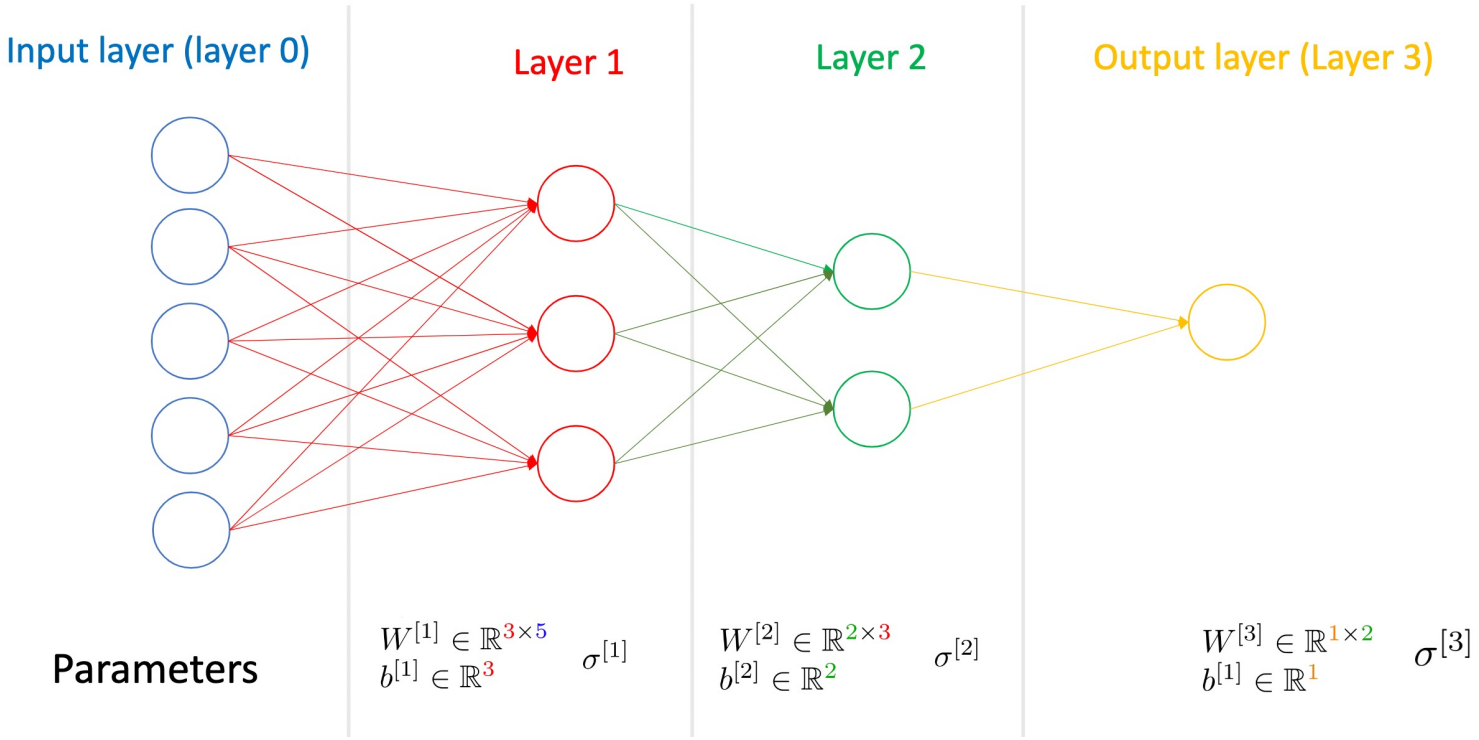
iii) *and an* **activation function** $\sigma : \mathbb{R}^m \to \mathbb{R}^m$.



r-layer Neural Network

Definition 6.3. *For $r \geq 1$ a* **r-layer neural network** *$N = (L^{[1]}, \ldots, L^{[r]})$ of* **input size** *n and* **output size** *m, consists of a collection of layers $L^{[i]} = (W^{[i]}, b^{[i]}, \sigma^{[i]})$ for $i = 1, \ldots, r$, such that $W^{[i]} \in \mathbb{R}^{m_i \times m_{i-1}}$ with $m_0 = n$ and $m_r = m$.*

The following shows an example of a 3-layer neural network with input size 5 and output size 1 and $m_0 = 5, m_1 = 3, m_2 = 2$, and $m_3 = 1$:



Neural Network: forward pass

Definition 6.4. *Let $N = (L^{[1]}, \dots, L^{[r]})$ be a r-layer neural network of input size n and output size m. We want to view it as a function $N : \mathbb{R}^n \to \mathbb{R}^m$ by defining $N(x)$ for an* **input** *$x \in \mathbb{R}^n$ by the output of its last layer $N(x) = a^{[r]}$. Here we define for $i = 1, \dots, r$ the following:*

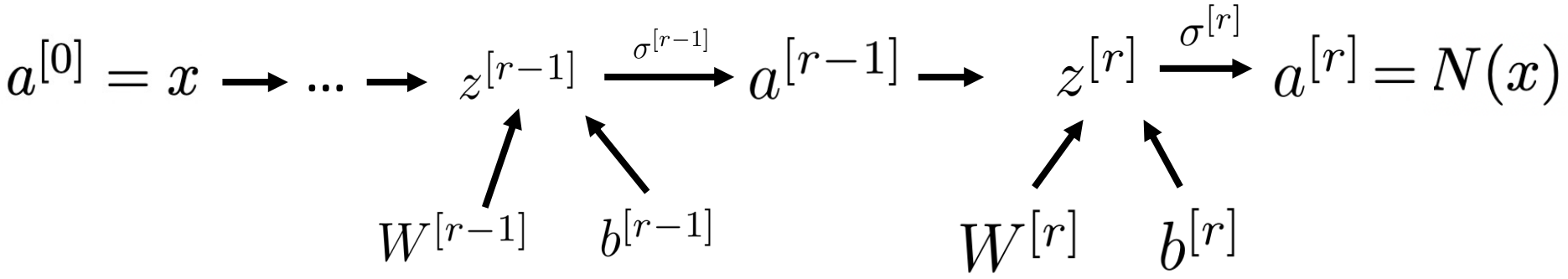
(i) The **linear part** *of the layer $L^{[i]}$ is defined by*

$$z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]},$$

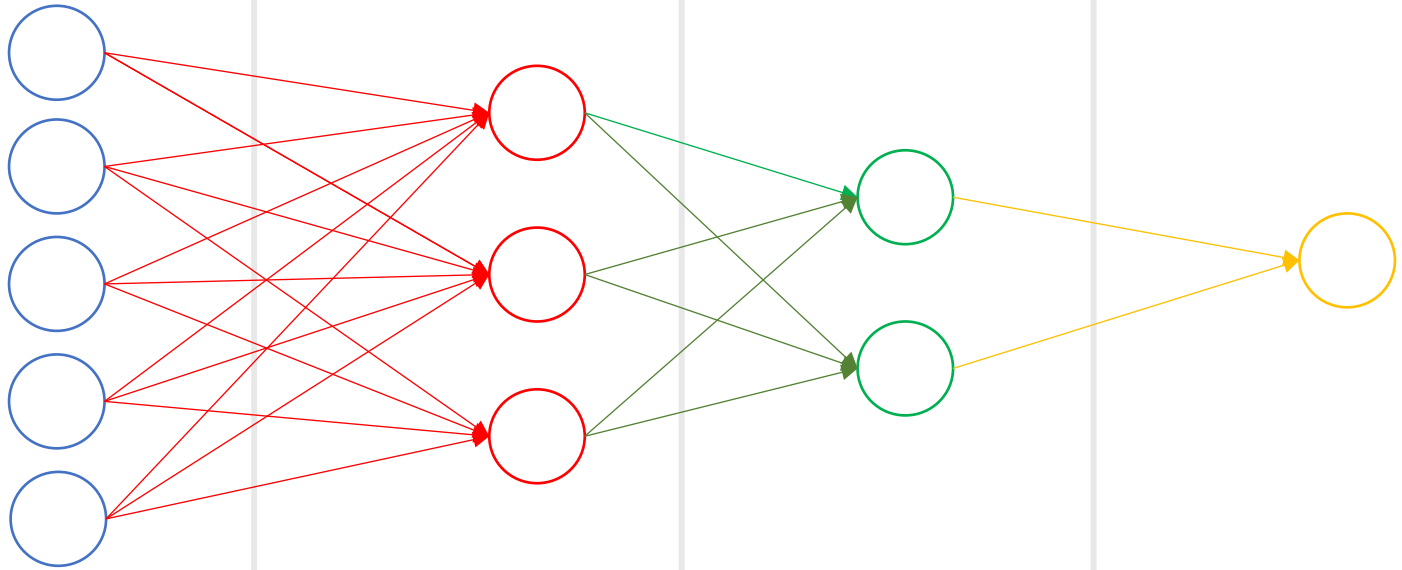
where $a^{[i-1]} \in \mathbb{R}^{m_{i-1}}$ is the output from the previous layer. In the case $i = 1$ we set $a^{[0]} = x$.

(ii) The **output** *of the layer $L^{[i]}$ is defined by applying the activation function to the linear part, i.e.*

$$a^{[i]} = \sigma^{[i]}(z^{[i]}) \in \mathbb{R}^{m_i}$$



Neural Network with 3 layers



Parameters

$W^{[1]} \in \mathbb{R}^{3\times 5}$
$b^{[1]} \in \mathbb{R}^{3}$
$\sigma^{[1]}$

$W^{[2]} \in \mathbb{R}^{2\times 3}$
$b^{[2]} \in \mathbb{R}^{2}$
$\sigma^{[2]}$

$W^{[3]} \in \mathbb{R}^{1\times 2}$
$b^{[1]} \in \mathbb{R}^{1}$
$\sigma^{[3]}$

Hidden layers

Training set

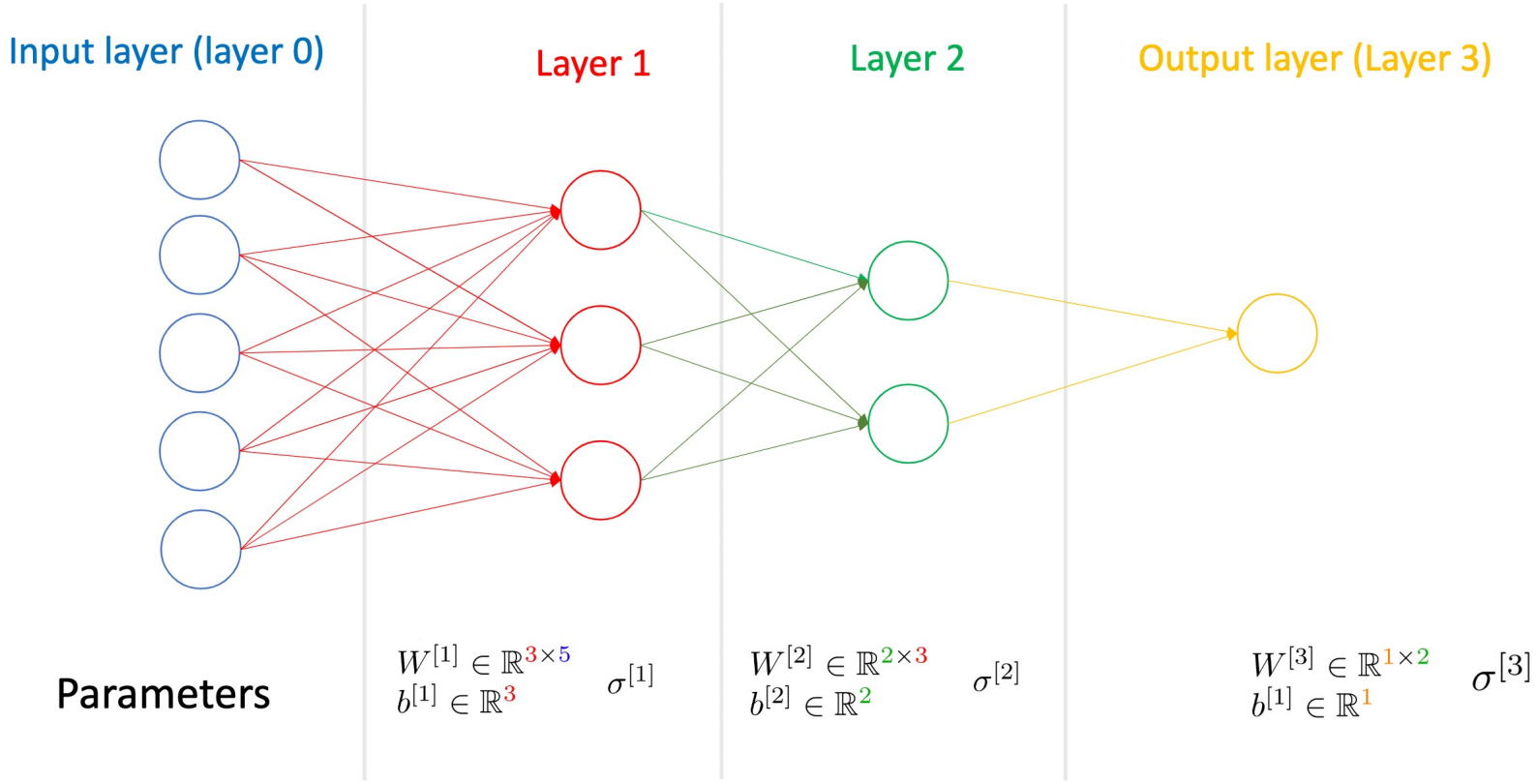
Our neural network is supposed to approximate a function $f : \mathbb{R}^n \to \mathbb{R}^m$(In the example $n = 5$ and $m = 1$). So we want to find the best possible choices of paramters (i.e. weight matrices and biases) such that $N(x)$ is a good approximation for $f(x)$.

- Input values (Feature space): $\mathcal{X} = \mathbb{R}^n$
- Output value (Label space): $\mathcal{Y} = \mathbb{R}^m$
- Training example: $(x, y) \in \mathcal{X} \times \mathcal{Y}$.
- Training set (with t training examples): $\mathcal{T} = \left((x^{(1)}, y^{(1)}), \ldots, (x^{(t)}, y^{(t)})\right) \in (\mathcal{X} \times \mathcal{Y})^t$.

Goal: Find a neural network N, such that N(x) is a good approximation for a given training set.

Strategy: Define a cost function and minimize it

Cost function in our example



An example of a neural network with $(3 \cdot 5 + 3) + (2 \cdot 3 + 2) + (1 \cdot 2 + 1) = 29$ parameters.

In our example we have 29 parameters, i.e. a cost function $J : \mathbb{R}^{29} \to \mathbb{R}$, which we want to minimize.

Recall: Cost functions

Given a training set $\mathcal{T}$, a **cost function** is a map from the space of parameters to $\mathbb{R}$, which measures how good the current parameters are with respect to the training set.

For linear regression we used the sum of squares:

$$J(\theta) = \frac{1}{2} \sum_{j=1}^{n} (h_\theta(x^{(j)}) - y^{(j)})^2 \, .$$

For logistic regression we used the log likelihood (logistic cost function)

$$J(\theta) = -\sum_{j=1}^{n} y^{(j)} \log h_\theta(x^{(j)}) + (1 - y^{(j)}) \log(1 - h_\theta(x^{(j)}))$$

Recall: Gradient descent

Gradient descent algorithm (rough version).

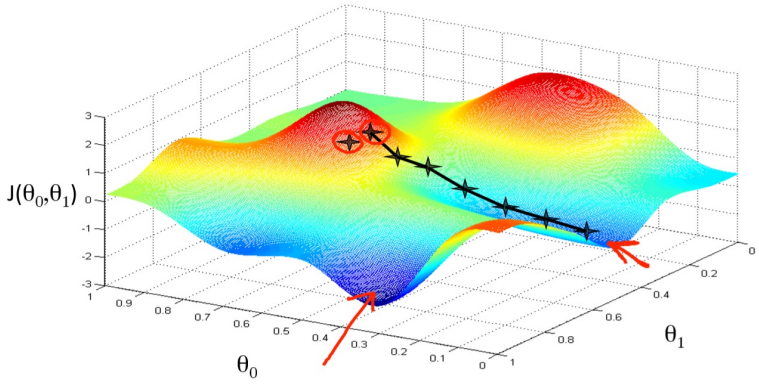
i) Start with a random starting value for the parameters, e.g. $\theta = 0 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$.

ii) Change the parameters in the opposite direction of the steepest ascent, i.e. opposite direction of the gradient. This means we want to subtract the gradient from the current parameters, weighted by a factor $\alpha \in \mathbb{R}$, the **learning rate**.

The new parameters θ are therefore given by:

$$\theta := \theta - \alpha \nabla J(\theta).$$

iii) Repeat step ii) until the value $J(\theta)$ does not change anymore.



Gradient descent in our example

In our example we have 29 parameters, i.e. a cost function $J : \mathbb{R}^{29} \to \mathbb{R}$, which we want to minimize.

We need to calculate the **gradient of** J

$$\nabla J = \begin{pmatrix} \frac{\partial}{\partial \theta_1} J \\ \frac{\partial}{\partial \theta_2} J \\ \vdots \\ \frac{\partial}{\partial \theta_{29}} J \end{pmatrix}$$

Main idea (**Backpropagation**): Calculate the gradient layer by layer using the chain rule

Backpropagation

Suppose we have a training set $\mathcal{T} = \left((x^{(1)}, y^{(1)}), \ldots, (x^{(t)}, y^{(t)})\right)$

For a neural network N, we will think of the collection of all weight matrices and biases as a vector $\theta \in \mathbb{R}^d$, called the **parameters**. For example, θ_1 could be the top left entry of the first weight matrix. The output of a neural network depends on the current choice of θ and therefore we write N_θ.

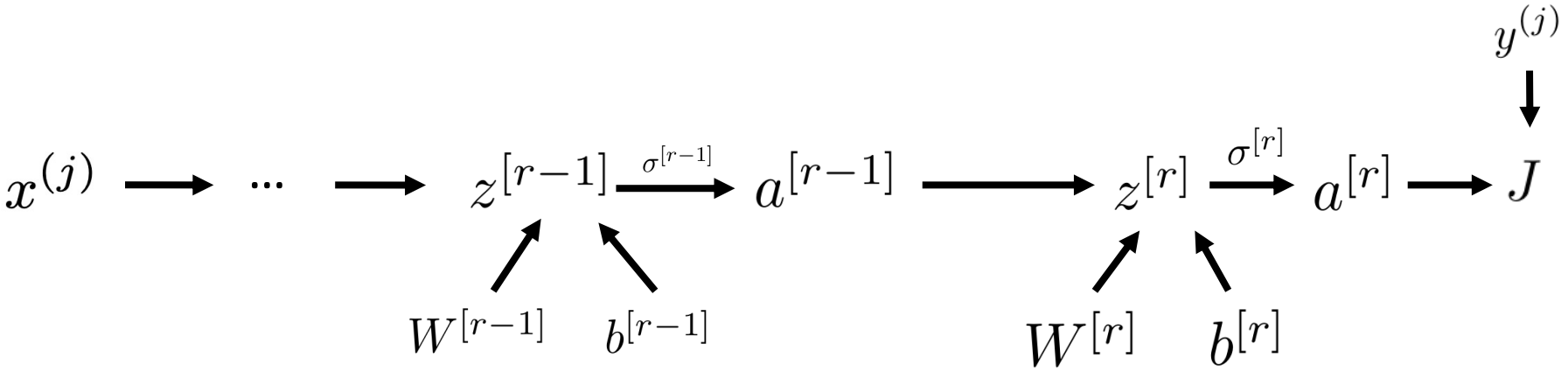
For example, we could use the sum of squares as a cost function

$$J(\theta) = \frac{1}{2} \sum_{j=1}^{n} \| N_\theta(x^{(j)}) - y^{(j)} \|^2 .$$

Backpropagation: A simple example in one dimensions

Backpropagation: General idea

Suppose we have a training set $\mathcal{T} = \left((x^{(1)}, y^{(1)}), \ldots, (x^{(t)}, y^{(t)})\right)$



$$z^{[r]} = W^{[r]} a^{[r-1]} + b^{[r]}$$

Derivative with respect to a matrix/vector

Backpropagation

1: Compute and store the values of $a^{[k]}$'s and $z^{[k]}$'s for $k = 1, \ldots, r$, and J.
. ▷ This is often called the "forward pass"
2: .
3: **for** $k = r$ to 1 **do** ▷ This is often called the "backward pass"
4: **if** $k = r$ **then**
5: compute $\delta^{[r]} \triangleq \frac{\partial J}{\partial z^{[r]}}$
6: **else**
7: compute

$$\delta^{[k]} \triangleq \frac{\partial J}{\partial z^{[k]}} = \left(W^{[k+1]^\top} \delta^{[k+1]} \right) \odot \mathrm{ReLU}'(z^{[k]})$$

8: Compute

$$\frac{\partial J}{\partial W^{[k]}} = \delta^{[k]} a^{[k-1]^\top}$$

$$\frac{\partial J}{\partial b^{[k]}} = \delta^{[k]}$$