UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI **MATEMATICA**
E INFORMATICA

ICAR CNR

Deep Learning

Carlo Adornetto

Laboratory 02 - Building Neural Network from Scratch

Credits: Angelica Liguori (ICAR)

UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI **MATEMATICA**
E INFORMATICA

Summarizing Neural Networks

A neural network is a tuple:

$$\textbf{net} = \{g, l, o, i, fpp\}$$

where:

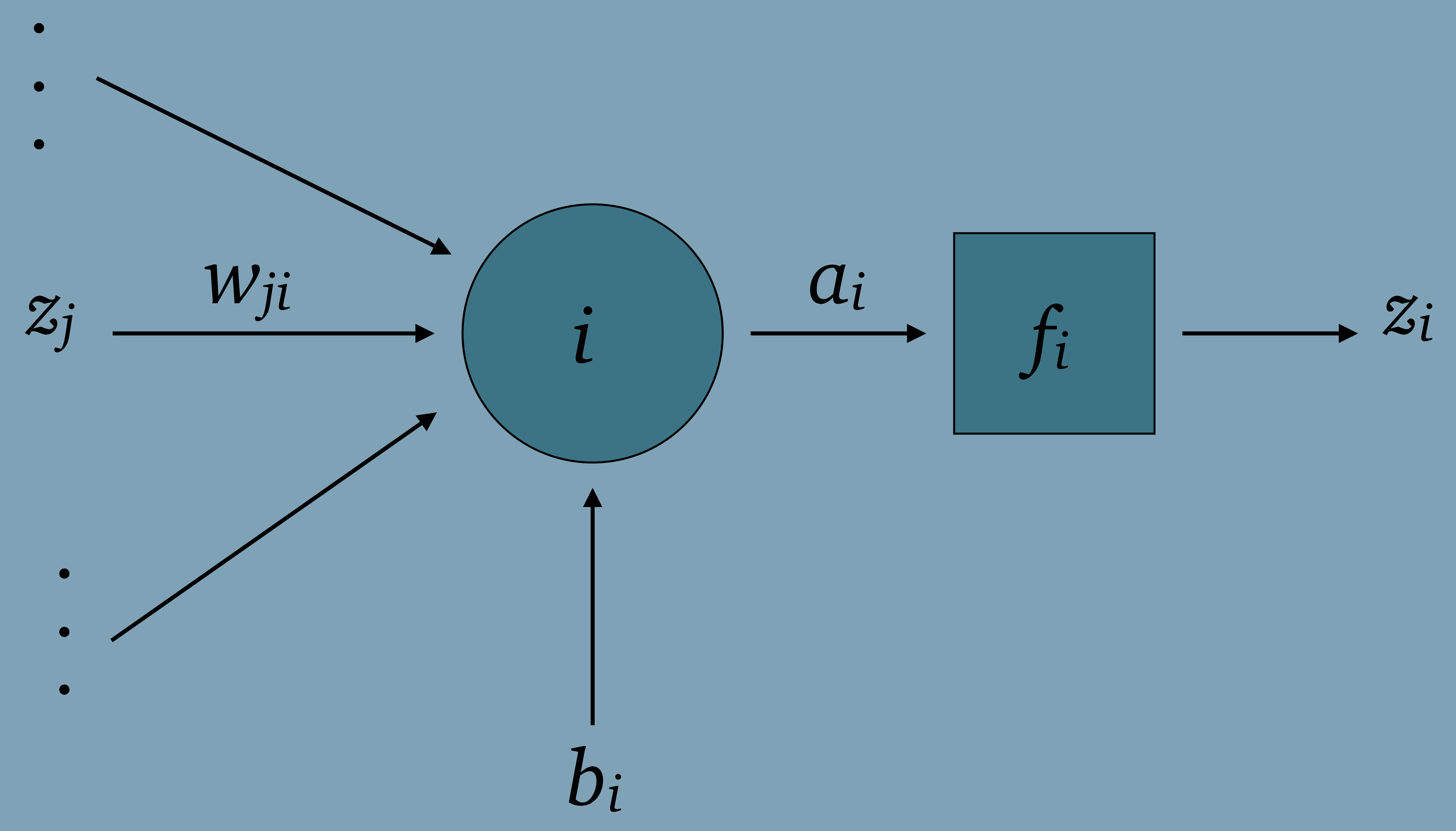
- *g* the graph
- *l* the loss function
- *o* the optimizer
- *i* the initialization
- *fpp* the *fix point procedure*

g the graph

- g = {N, E} is a weighted labeled directed graph
- Each node $i \in N$ is also called *neuron* or *perceptron*
  - It is equipped with two labels
    - A value a_i that will called ***activation*** in the next
    - An activation function f_i that, applied to the activation, produces an output z_i
- Each edge $e = \{j \rightarrow i\} \in E,\ i,j \in N,$ is equipped with a weight w_{ji}
- Each node i is involved in an additional special edge, with a ghost node, which weight is called bias (b_i)

g the graph

- Each neuron is a calculus unit

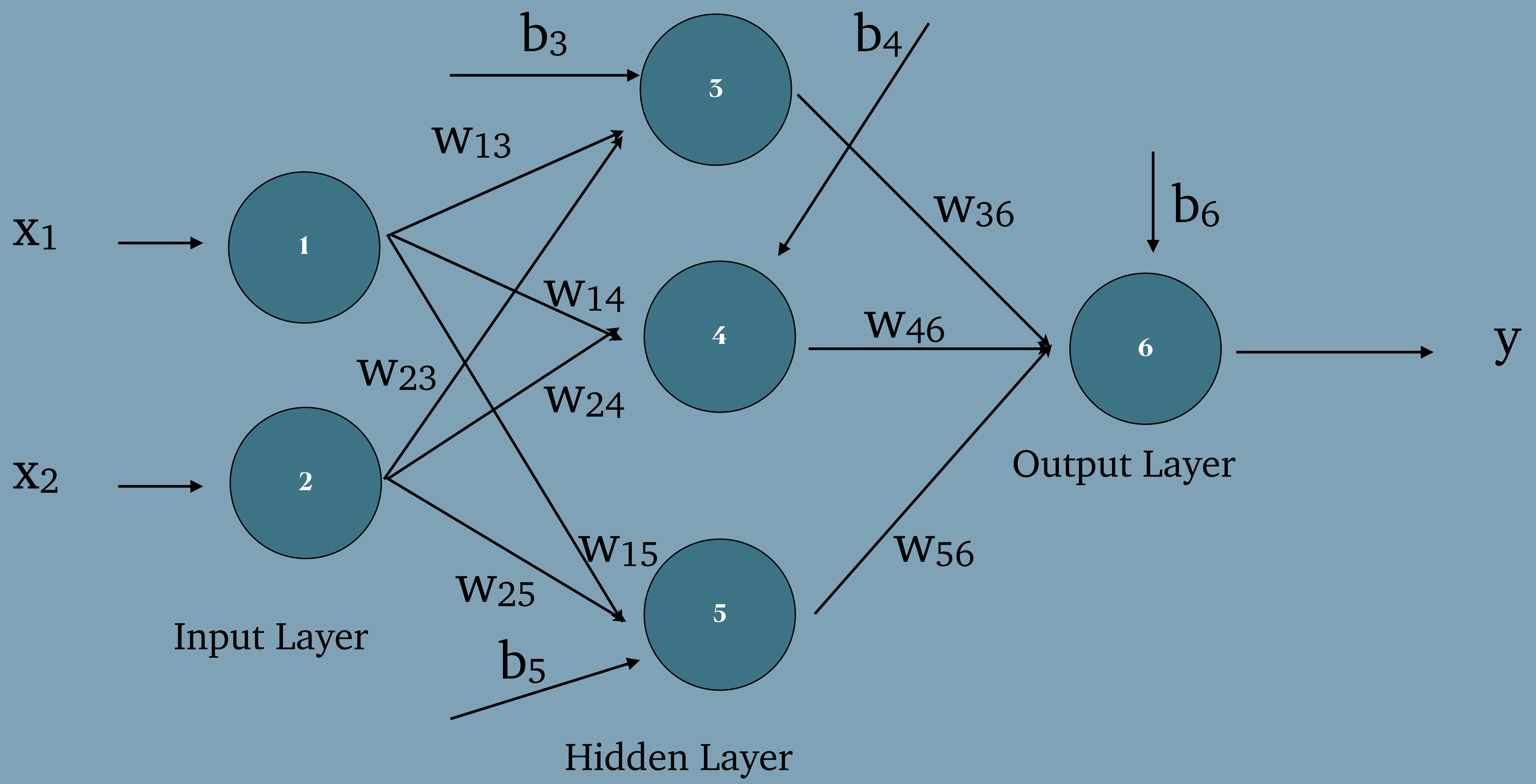


$$z_i = f_i(a_i)$$

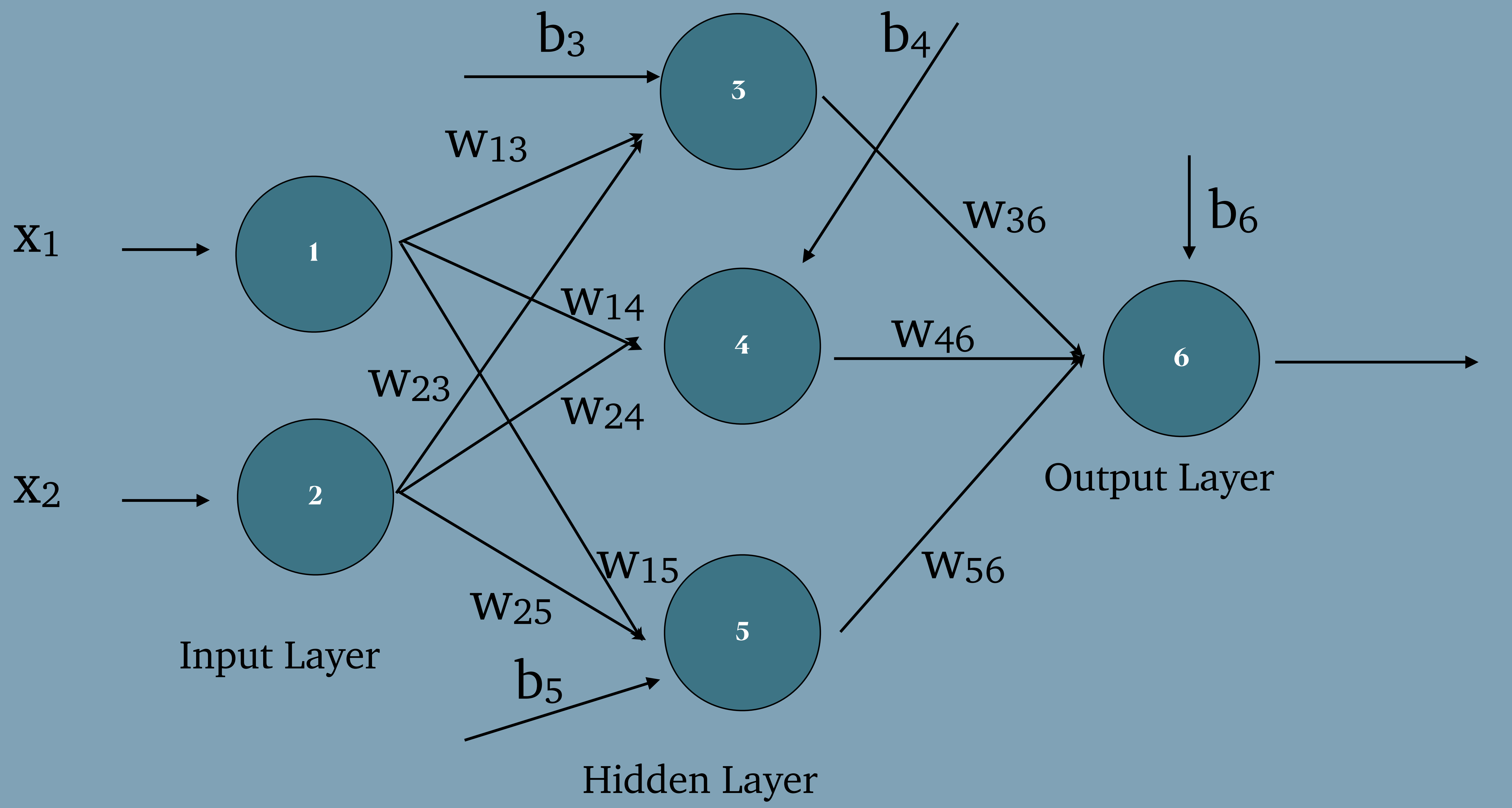
$$a_i = b_i + \sum_{j:j \to i \in E} w_{ji} z_j$$

g the graph

- A combination of connected neurons builds the graph up
- Nodes that share the same input are grouped into layers



g the graph



- For the nodes that belong at the input layer, the output z_i is equal to the input.

$$z_1 = x_1$$

$$z_2 = x_2$$

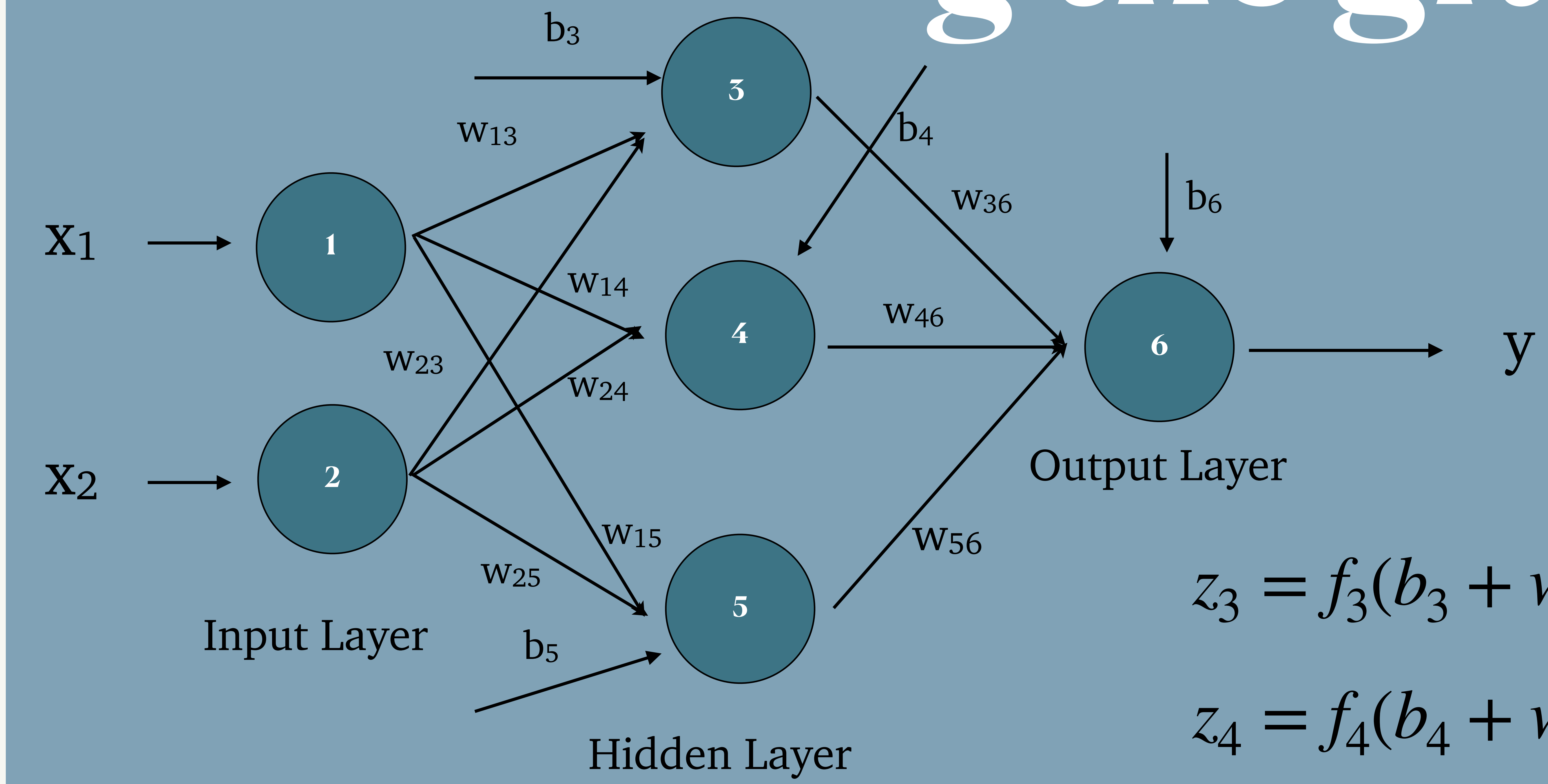
$$z_3 = f_3(b_3 + \sum_{j:j\to 3\in E} w_{j3}z_j)$$

$$z_4 = f_4(b_4 + \sum_{j:j\to 4\in E} w_{j4}z_j)$$

$$z_5 = f_5(b_5 + \sum_{j:j\to 5\in E} w_{j5}z_j)$$

$$y = z_6 = f_6(b_6 + \sum_{j:j\to 6\in E} w_{j6}z_j)$$

g the graph



$$z_1 = x_1$$

$$z_2 = x_2$$

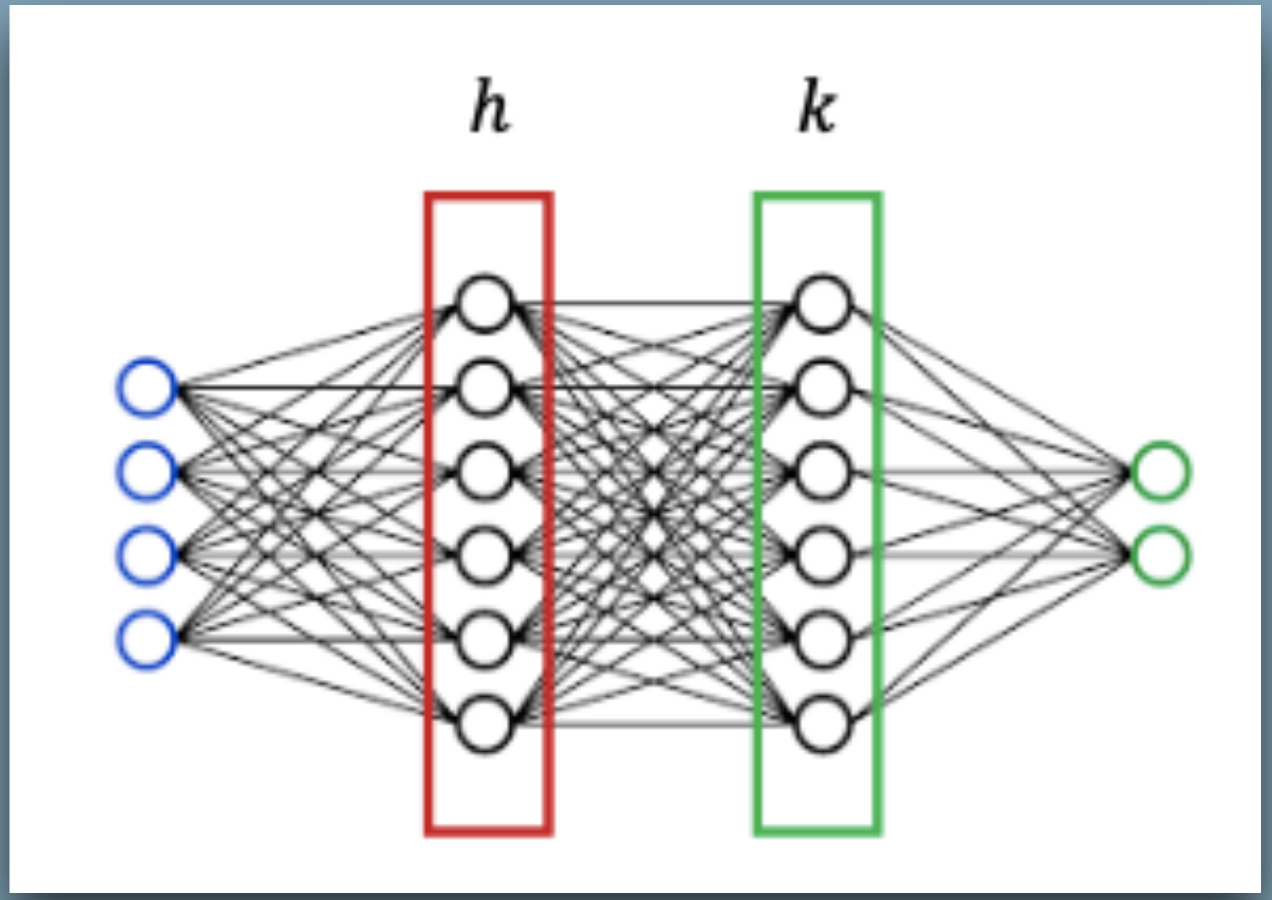
$$z_3 = f_3(b_3 + w_{1,3}z_1 + w_{2,3}z_2) = f_3(b_3 + w_{1,3}x_1 + w_{2,3}x_2)$$

$$z_4 = f_4(b_4 + w_{1,4}z_1 + w_{2,4}z_2) = f_4(b_4 + w_{1,4}x_1 + w_{2,4}x_2)$$

$$z_5 = f_5(b_5 + w_{1,5}z_1 + w_{2,5}z_2) = f_5(b_5 + w_{1,5}x_1 + w_{2,5}x_2)$$

$$y = z_6 = f_6(b_6 + w_{3,6}z_3 + w_{4,6}z_4 + w_{5,6}z_5) = f_6(b_6 + w_{3,6}f_3(b_3 + w_{1,3}x_1 + w_{2,3}x_2) +$$
$$+ w_{4,6}f_4(b_4 + w_{1,4}x_1 + w_{2,4}x_2) + w_{5,6}f_5(b_5 + w_{1,5}x_1 + w_{2,5}x_2))$$

g the graph



- Compact notation:
  - Given two consecutive layers k and h:
  - $\vec{z}_k = f_k(\overrightarrow{b_k} + W_k \overrightarrow{z_h})$
- We are assuming that all the nodes in k share the same activation function f_k
- $\overrightarrow{b_k}$ contains all the biases of the nodes in k
- W_k is the matrix containing all the $w_{h,k}$ weights

/ the loss function

- The graph is actually a non linear algebraic operator

- The operator is composed by a priori unknown variables:
  - The weights W and the biases B

- The learning phase of a neural network aims at finding the “best” values for W and B

/ the loss function

- What does “best” mean?

- Finding the “best” values needs to optimize an objective function that expresses the semantics of the analysis goals
  - For what purpose are we using the neural networks?
  - What is the input?
  - What is the desired output?
  - How far is the produced output from the desired output?

/ the loss function

- In neural networks the objective function is called *loss function* and it should be minimized
- The loss function represents the error in producing an output as close as possible to the desired one, by applying its operator *g* on the output
- The objective of a network is:

$$\arg\min_{W,B} \frac{1}{n} \sum_{i=1}^{n} loss[\overrightarrow{y_i}, g(\overrightarrow{x_i} \mid W, B)]$$

o the optimizer

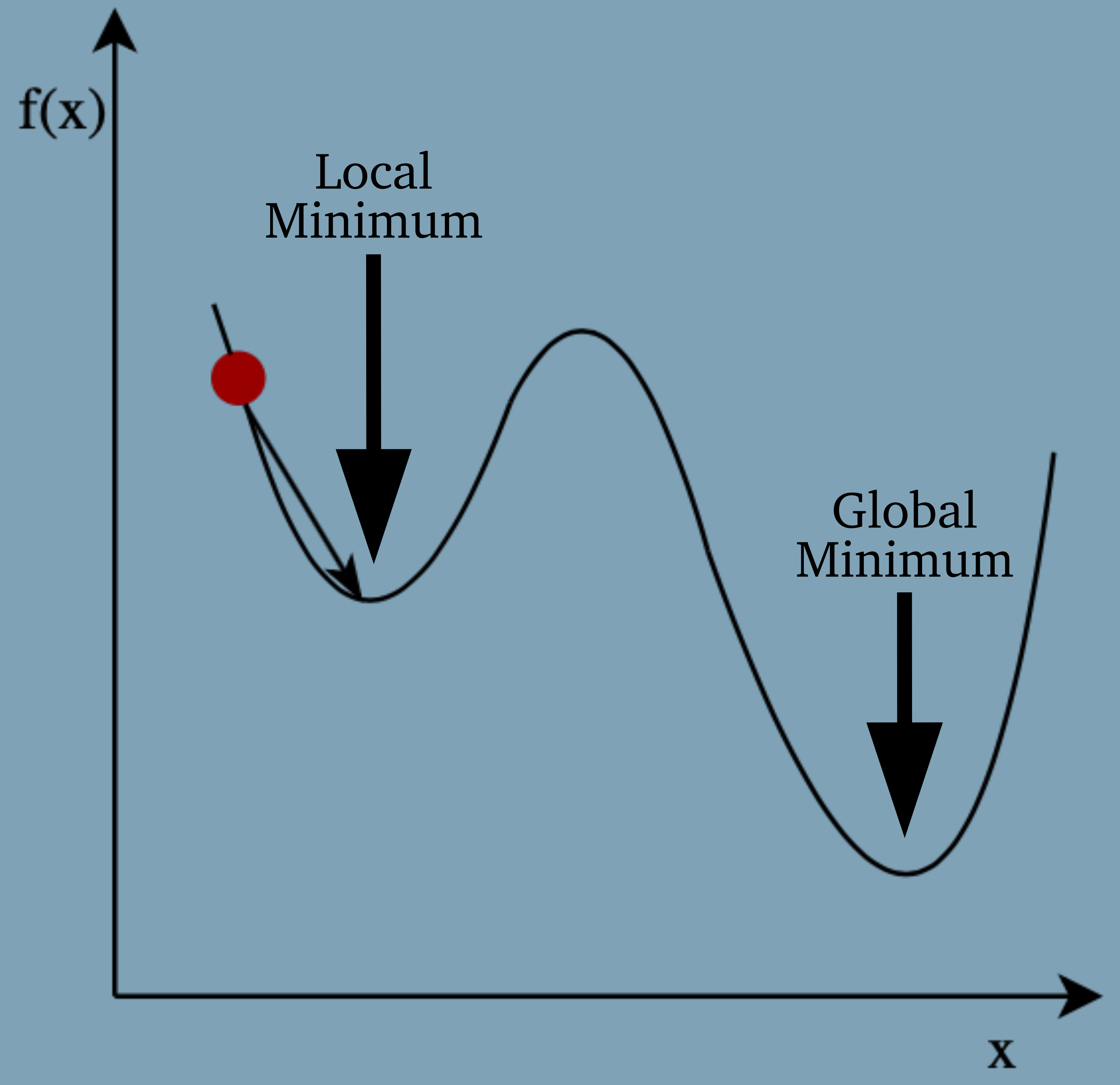
- How to solve this problem?

$$\arg\min_{W,B} \frac{1}{n} \sum_{i=1}^{n} loss[\overrightarrow{y_i}, g(\overrightarrow{x_i} \mid W, B)]$$

  - We can compute the gradient of the loss function
  - Put it equal to zero and
  - Check if the solutions are minima, maxima or saddle points

o the optimizer

- Dealing with a lot of (noisy) data and parameters makes hard to find an analytical solution
- We need to define an approximation, an heuristic
  - We have to be content with optimal (non optima) solutions
    - Gradient Descent (Finding local minima)



Gradient Descent

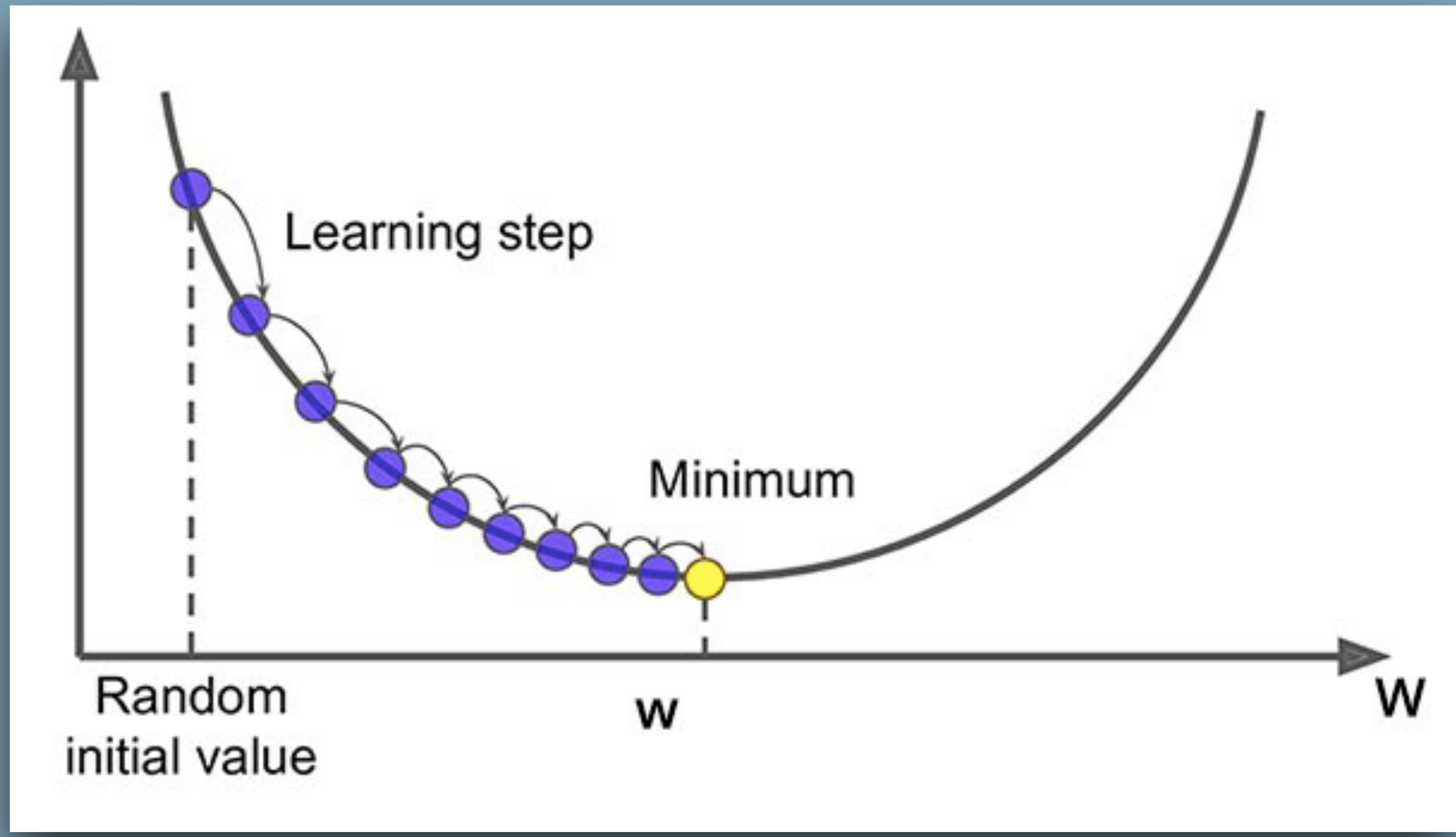
- Gradient Descent is an iterative algorithm for finding a local minimum of a differentiable function

- Let $F(\bar{x})$ be a multivariate and differentiable in a neighborhood of a point $\bar{a}$

    - $F(\bar{x})$ decreases fastest if one goes from $\bar{a}$ in the direction of the negative gradient

    - The algorithm is: update $\bar{a}$ until convergence:

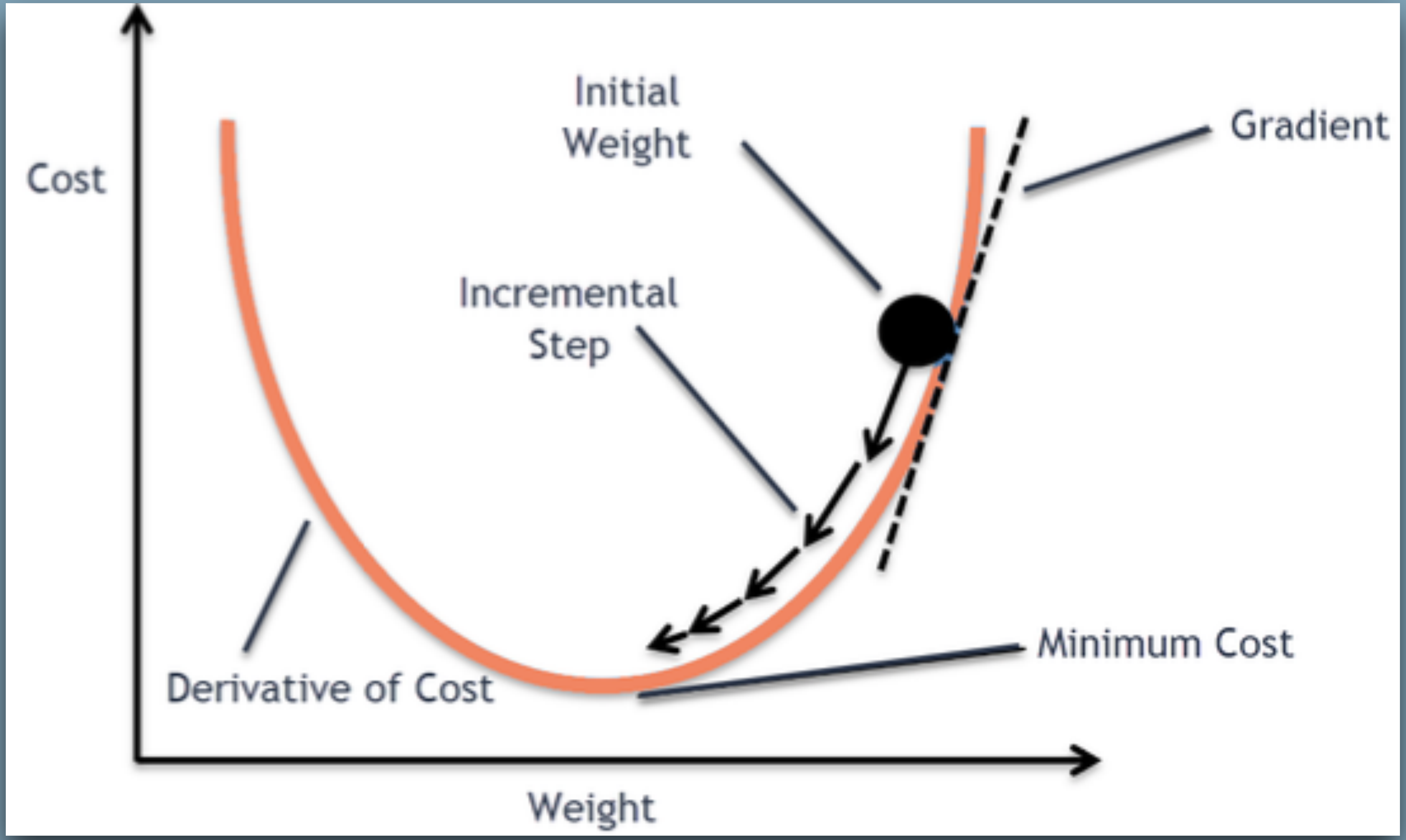
$$\bar{a}^{new} = \bar{a}^{old} - \eta \nabla F(\bar{a}^{old})$$



- The parameter η is called *learning rate* and determines the behavior of the optimization

Gradient Descent

- How does it work?
  - Start from a random point of the function to optimize
  - Compute the gradient on that point
  - Follow the gradient to find another point of the function that is closer to the optimum
  - Repeat until convergence



Gradient Descent

$$[W, B]_{t+1} = [W, B]_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{W,B} loss[\overrightarrow{y_i}, g(\overrightarrow{x_i} \mid W, B)]$$

- Where:
  - $[W, B]$ is the concatenation of the unknown parameters
  - $\nabla_{W,B}$ is the gradient operator
  - t is the convergence step
  - η is the *learning rate*
    - A term that decides how much we need to navigate the gradient

Backpropagation

- In a neural network, we are able to compute the $\nabla loss[\overrightarrow{y_i}, g(\overrightarrow{x_i} \mid W^*)]$ only in the last layer
  - It is the only one connected to the ground truth $\overrightarrow{y_i}$
- But we need to update all the weights
- Each layer $k \in \{1,\ldots,K\}$ has a weight matrix (with biases) W_k^* that contributes to the gradient

$$\nabla loss(\overrightarrow{y_i}, g(\overrightarrow{x_i} \mid W^*)) = \nabla loss(\overrightarrow{y_i}, f_K(W_K^* f_{K-1}(W_{K-1}^* f_{K-2}(W_{K-2}^* f_{K-3}(\ldots)))))$$

Backpropagation

- Now we can distribute the gradient on the layers
  - Each layer k contributes to the gradient:

$$\delta_k \equiv W^*_{k+1} f'_k(\ldots) W^*_{k+2} f'_{k+1}(\ldots) \ldots W^*_K f'_{K-1}(\ldots) loss'(\overrightarrow{y_i}, f_K(\ldots))$$

- But $\delta_k(W^*_k f'_{k-1}(\ldots))$ corresponds to $\nabla_{w^*_k} loss(\overrightarrow{y_i}, g(\overrightarrow{x_i} \mid W^*))$
- Then …

Backpropagation

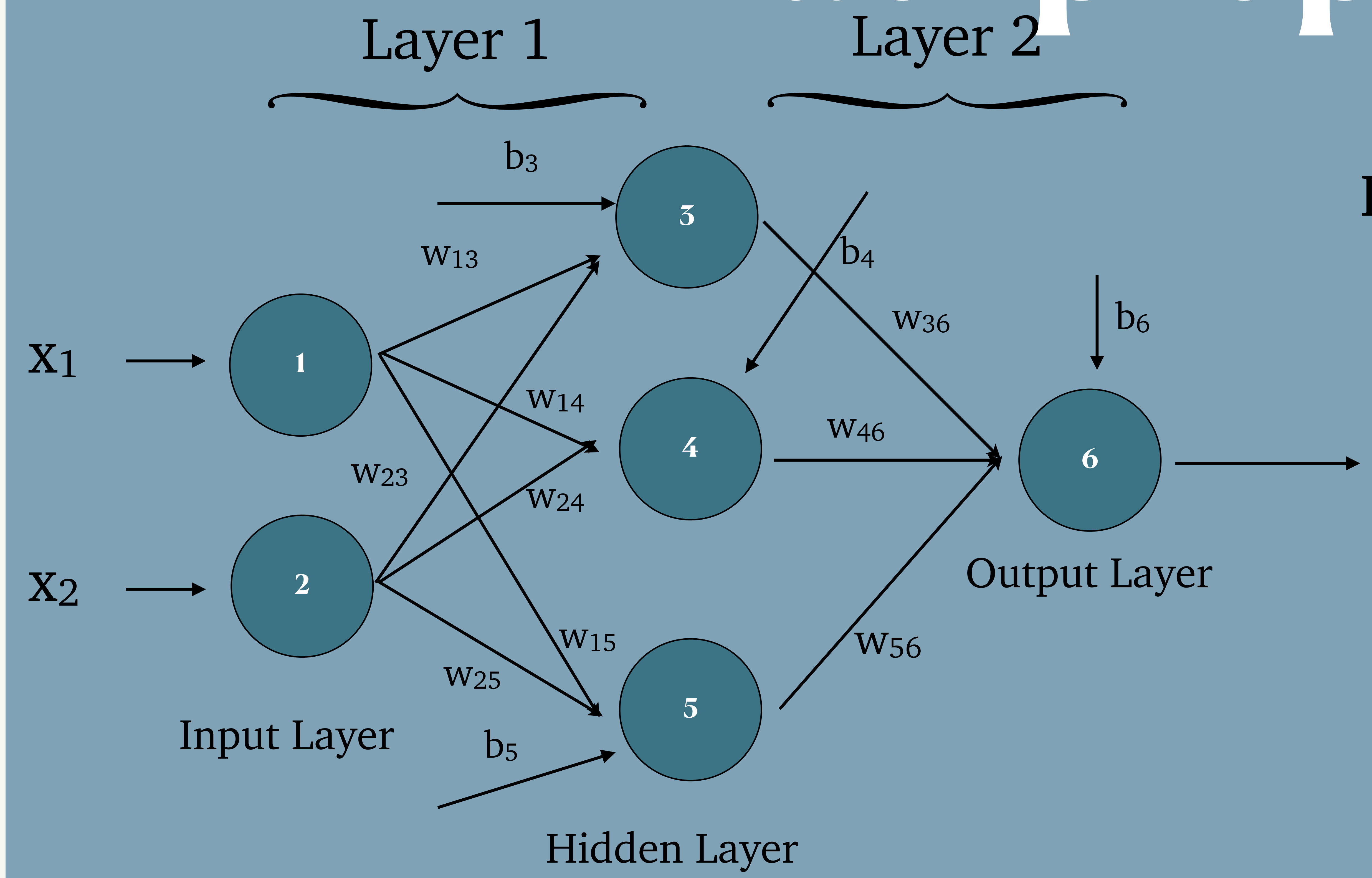
- … iterative computation

$$\delta_k = W^*_{k+1} f'_k(\ldots)\delta_{k+1}$$

- Since $\delta_k W^*_k f'_{k-1}(\ldots)$ corresponds to the contribution of the layer to the gradient, we can update each layer backwardly:

$$W^*_{k;t+1} = W^*_{k;t} - \eta \delta_k W^*_k f'_{k-1}(\ldots)$$

Backpropagation



Layer 2

$$\delta W_{36} = \underbrace{loss'(z_6, y) * f_6'(z_6)}_{\delta} * z_3$$

$$\delta W_{46} = (z_6 - y) * f_6'(z_6) * z_4$$

$$\delta W_{56} = (z_6 - y) * f_6'(z_6) * z_5$$

$$\delta W_{13} = \underbrace{(z_6 - y) * f_6'(z_6) * w_{36} * f_3'(z_3)}_{\delta_3} * z_1$$

Layer 1

$$\delta W_{14} = (z_6 - y) * f_6'(z_6) * w_{46} * f_4'(z_4) * z_1$$

$$\delta W_{15} = (z_6 - y) * f_6'(z_6) * w_{56} * f_5'(z_5) * z_1$$

$$\delta W_{23} = (z_6 - y) * f_6'(z_6) * w_{36} * f_3'(z_3) * z_2$$

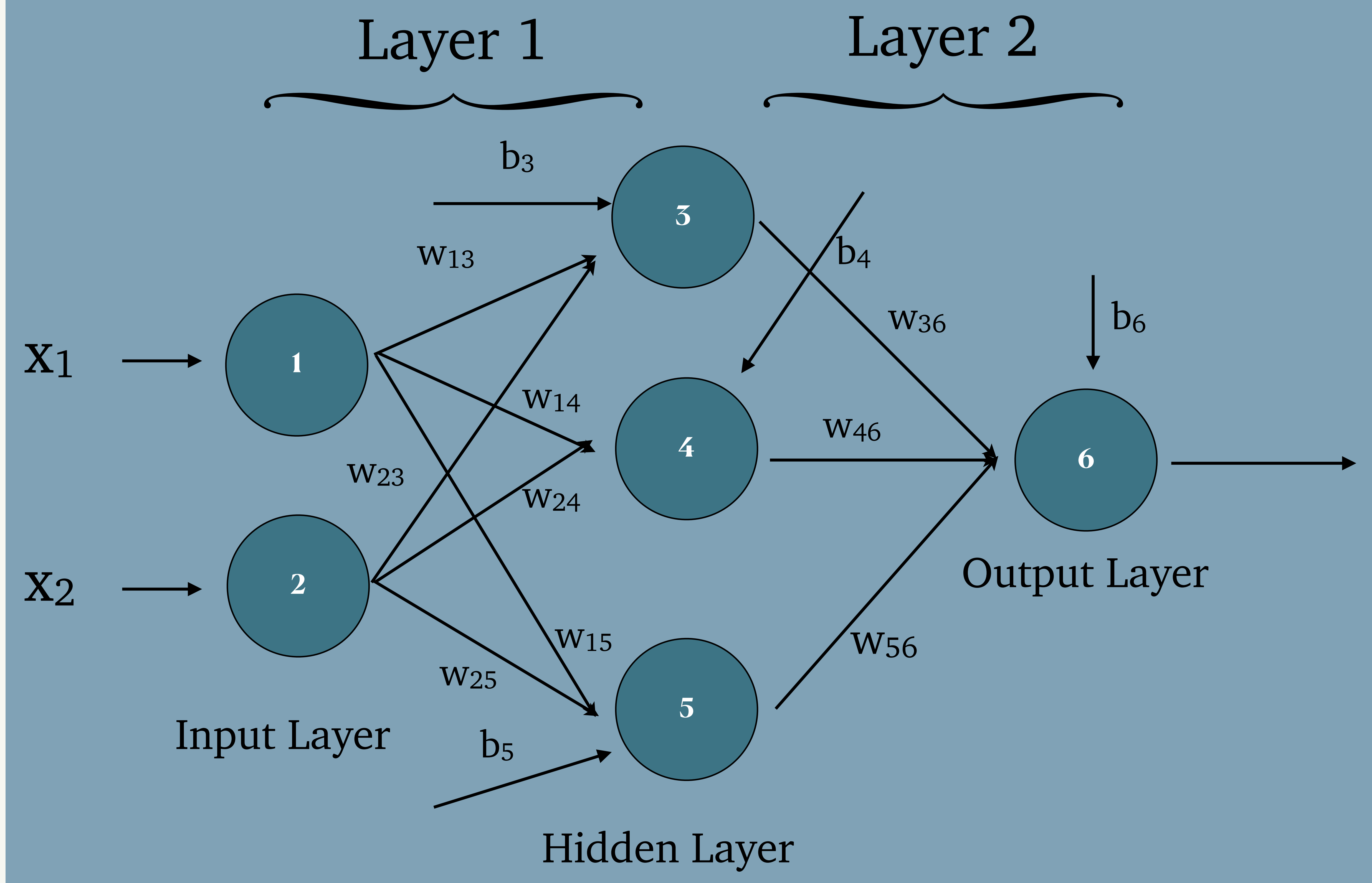
$$\delta W_{24} = (z_6 - y) * f_6'(z_6) * w_{46} * f_4'(z_4) * z_2$$

$$\delta W_{25} = (z_6 - y) * f_6'(z_6) * w_{56} * f_5'(z_5) * z_2$$

- Suppose that the loss is:

$$\frac{1}{2}\sum_i (y_i - z_i)^2$$

Backpropagation



Layer 2
$$\begin{cases} \delta W_2 = \underbrace{loss'(z_6, y) * f_2'(z_2)}_{\delta_2} * z_1 \\ \delta B_2 = \delta_2 \end{cases}$$

Layer 1
$$\begin{cases} \delta W_1 = \underbrace{(z_6 - y) * f_2'(z_2) * w_2 * f_1'(z_1)}_{\delta_1} * z_0 \\ \delta B_1 = \delta_1 \end{cases}$$

$$W_1 = W_1 - \eta * \delta W_1$$
$$W_2 = W_2 - \eta * \delta W_2$$
$$B_1 = B_1 - \eta * \delta B_1$$
$$B_2 = B_2 - \eta * \delta B_2$$

- Suppose that the loss is:

$$\frac{1}{2} \sum_i (y_i - z_i)^2$$

i the initialization

- In Gradient Descent, edges weights and biases need an **initial value**
- The initialization strategy may strongly change the network behavior
- Since we are searching for optimal solution, the **starting point** of the fix point procedure (Gradient Descent) is **crucial**

i the initialization

- Zero initialization
  - Bad solution
    - All the nodes have the same initial gradient
    - There is no diversification of the nodes
      - Hidden nodes becomes symmetric
- Constant initialization?
  - It has the same problems

i the initialization

- This means we need different values
- But keep in mind:
  - If weights are initialized with very high values, asymptotical activation functions (e.g. sigmoid, tanh) produce a gradient that is practically equal to 0
    - Low gradient → learning takes a lot of time
  - If weights are initialized with low values it gets mapped to 0, where the case is the same as before
- Simplest initialization
  - Random $W_k \sim Uniform()$
  - Random $W_k \sim N(0,1)$

fpp the fix point procedure

- Training a neural network is a simple procedure
  - We are search for a fix point as loss optimal solution
- The algorithm

```
net = CustomNeuralNetwork(...)
initialize_weights_and_biases(net)
optimizer = myOptimizer(...)
loss_function = myLossFunction(...)

epochs = ... # the number of dataset scans
history = [] # a list containing the loss evolution

for epoch in range(epochs):
  optimizer.reset() # it may have an internal status
  loss = loss_fuction(out_target, net(input))
  back_propagation(loss, optimizer, net)

  history.append(loss)
```