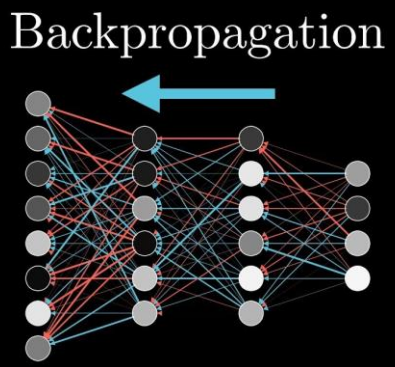
Lecture 4 : NN

By : Khalil idrissi

- History ☹
- A simple neural net : FFNN
- Activation functions
- Cost functions
- Backpropagation
- Optimizers
- Weight initialization
- Techniques to avoid overfitting



To be a good person, you have to learn the 3 of them

Pytorch

Keras

Tensorflow

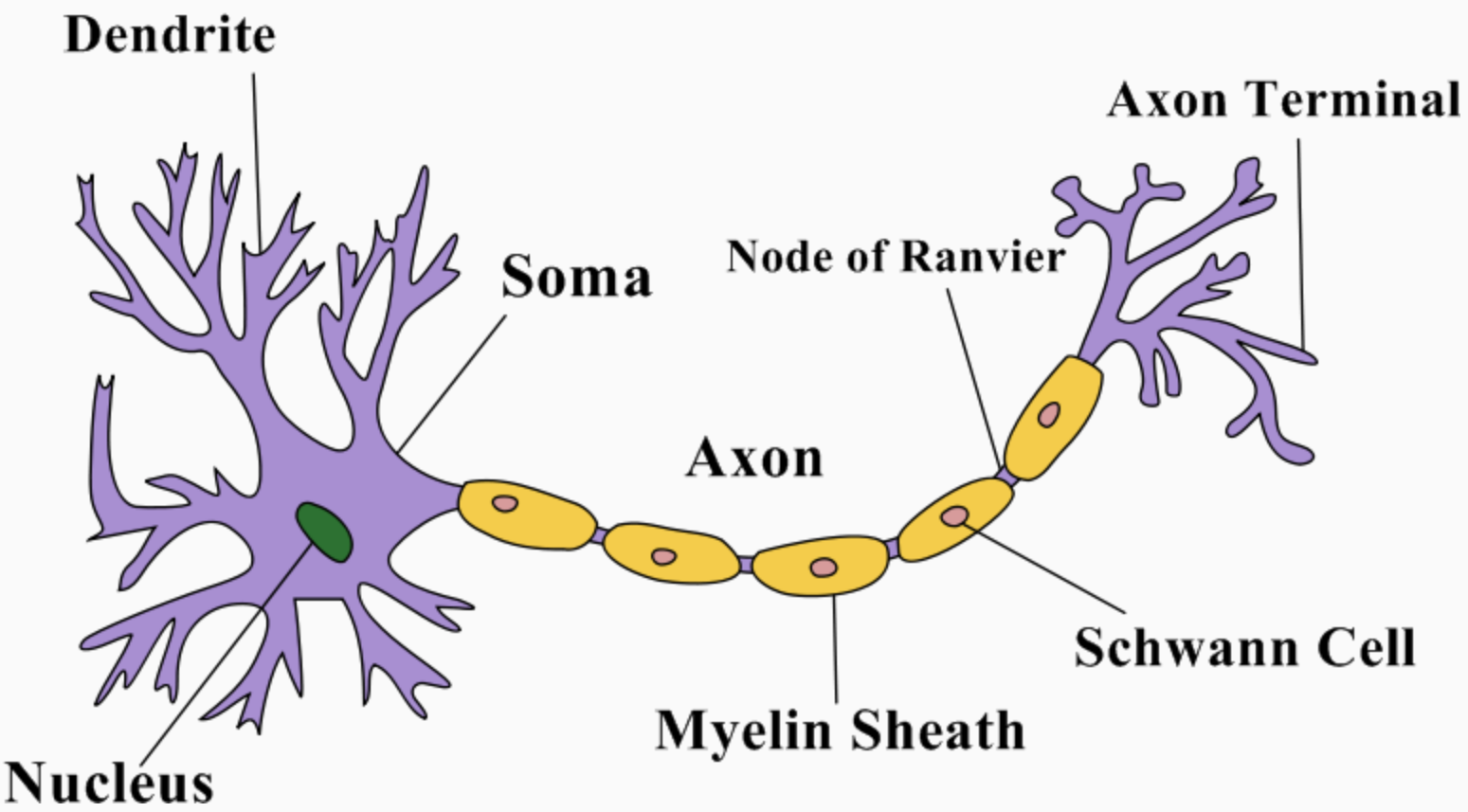
Pytorch

Tensorflow

In this course we will focus on:

Pytorch

Schematic Diagram of a Biological Neuron :



1943

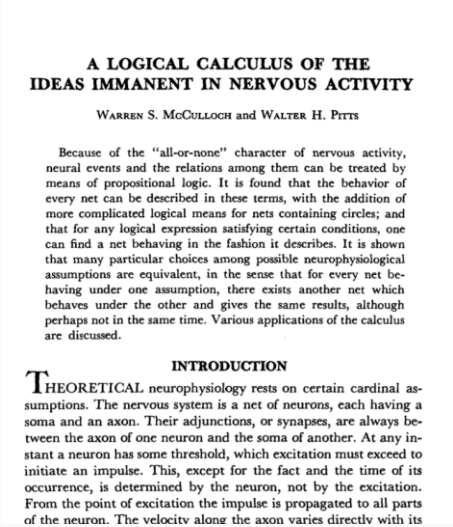
1943

Warren McCulloch
1898-1969

Walter Pitts
1923-1969

1943

Warren McCulloch
1898-1969



A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

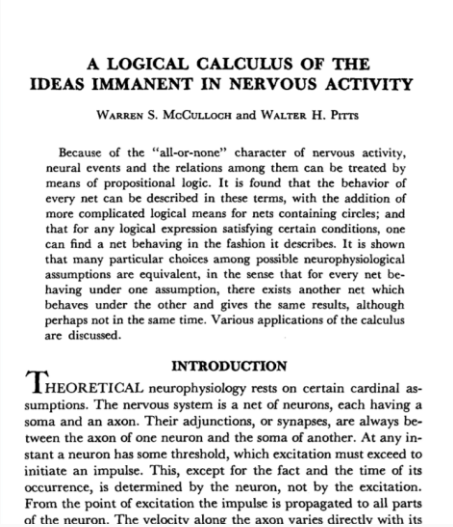
INTRODUCTION

THEORETICAL neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurrence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its

Walter Pitts
1923-1969

1943

Warren McCulloch
1898-1969



A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS

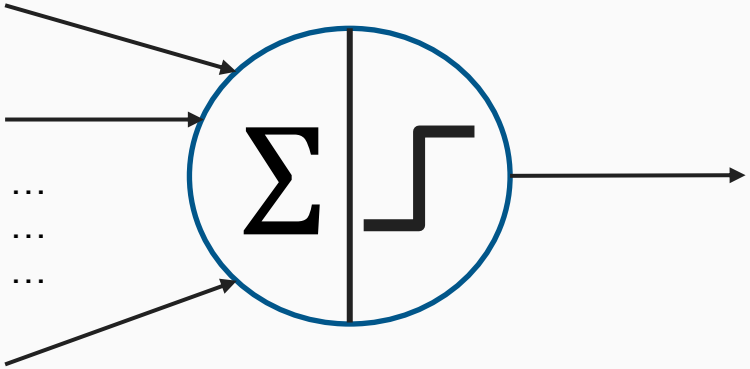
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

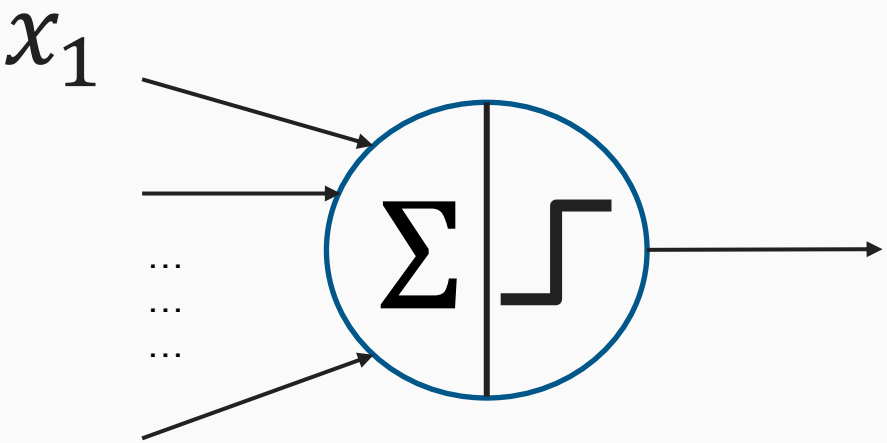
INTRODUCTION

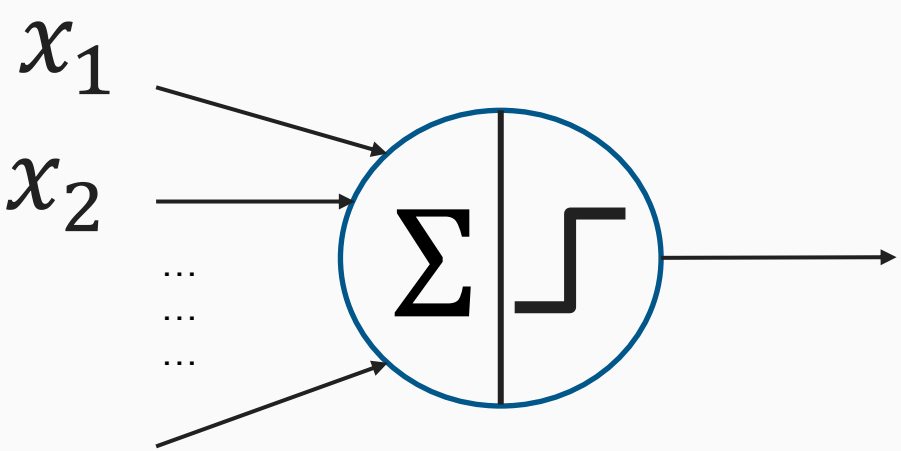
THEORETICAL neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurrence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its

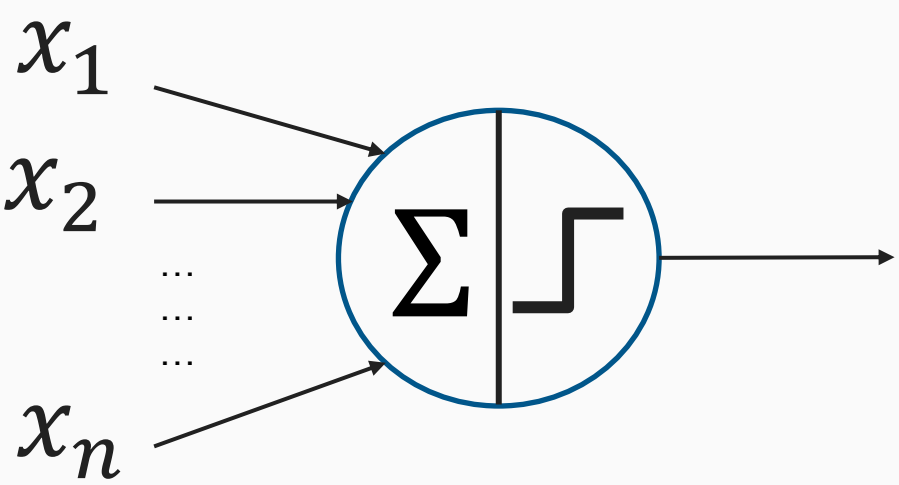
Walter Pitts
1923-1969

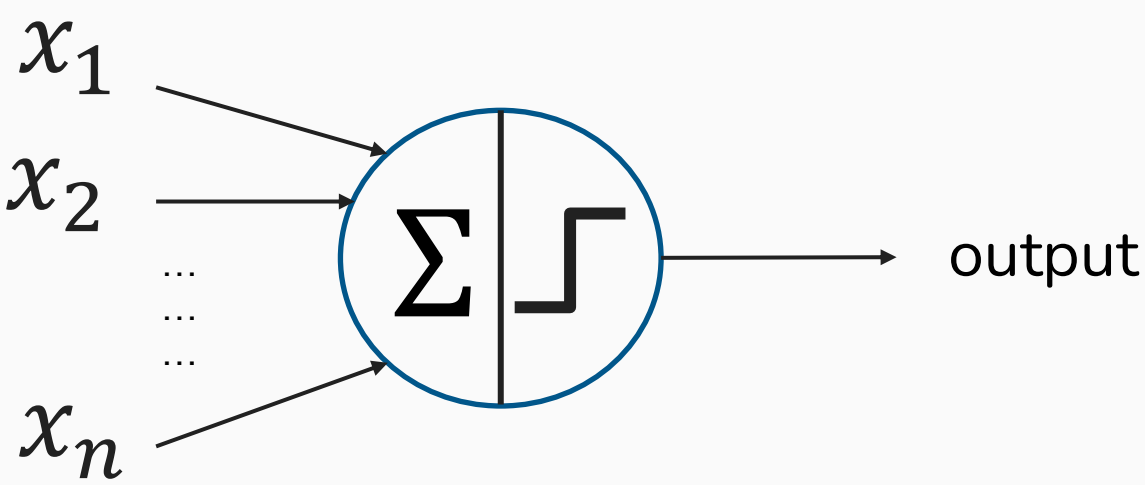
modeled a simple neural network using electrical circuits in order to describe how neurons in the brain might work

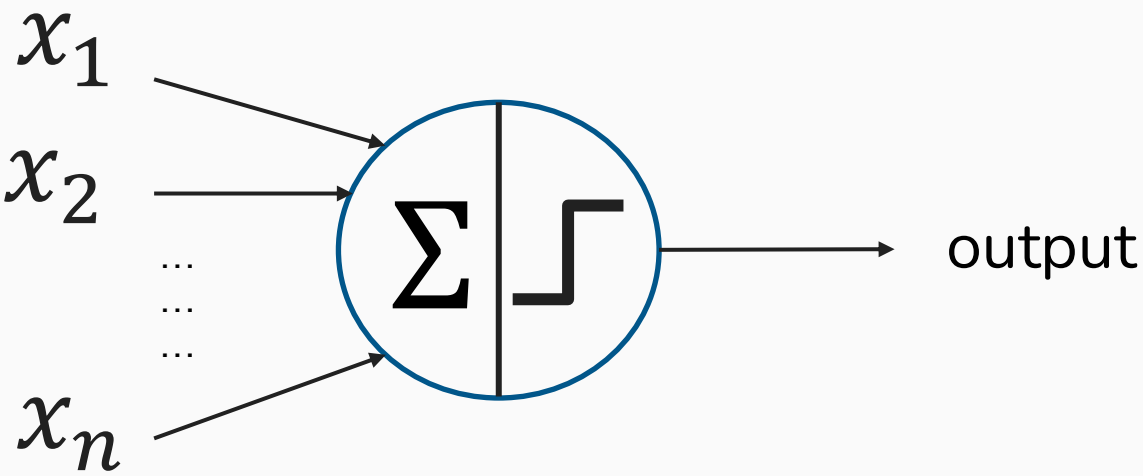



x_1
...
...
...
Σ

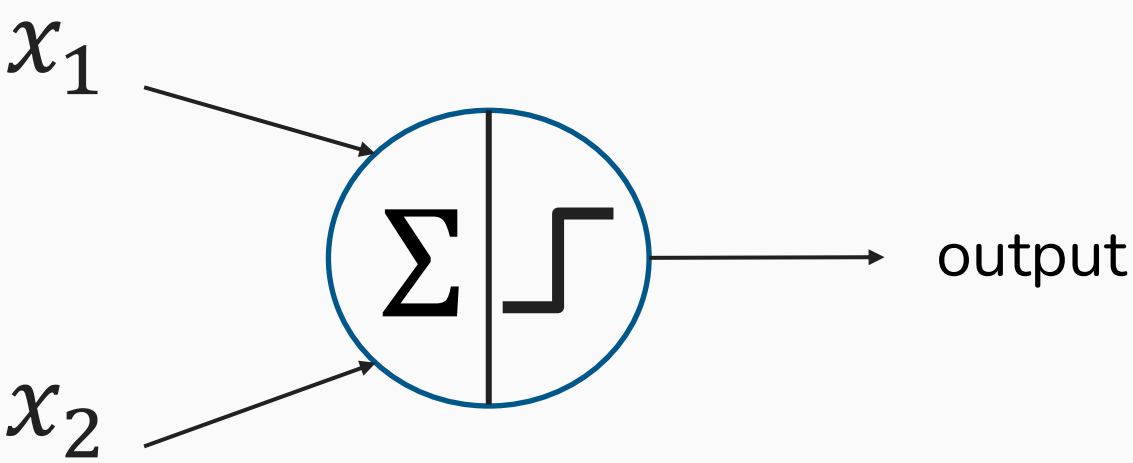

x_1
x_2
...
...
...
Σ


x_1
x_2
...
...
...
x_n
Σ


x_1
x_2
...
...
...
x_n
Σ
output

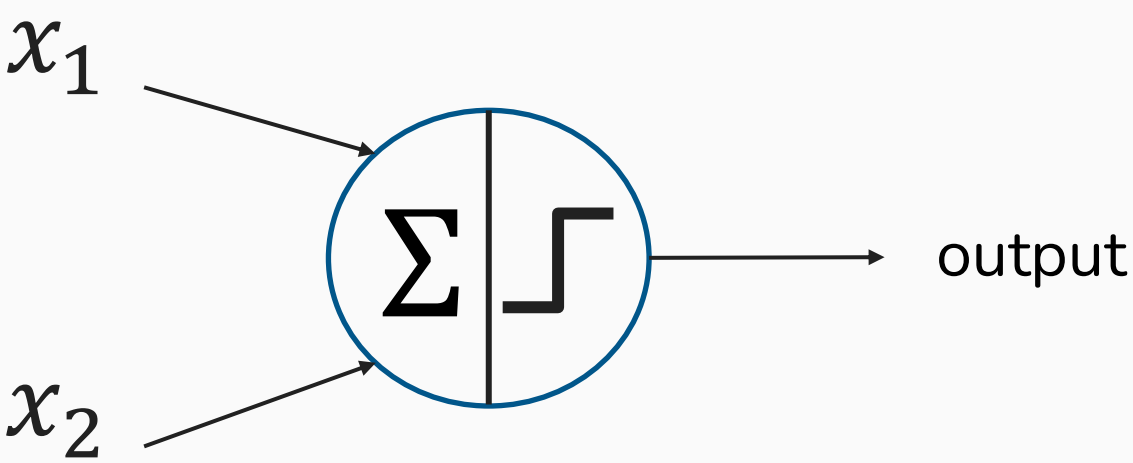


With a threshold of 2.0 they modeled and problem



With a threshold of 2.0 they modeled and problem

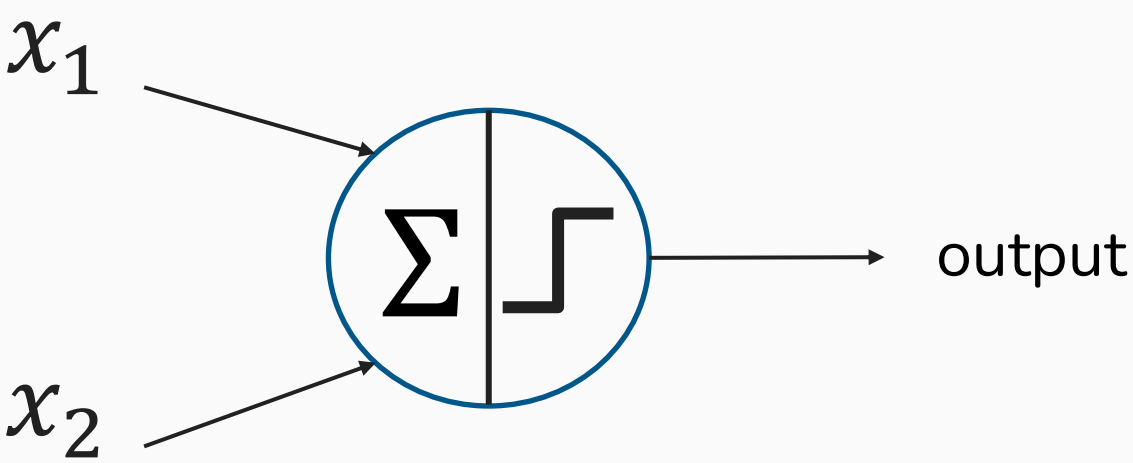
x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1



With a threshold of 2.0 they modeled and problem

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

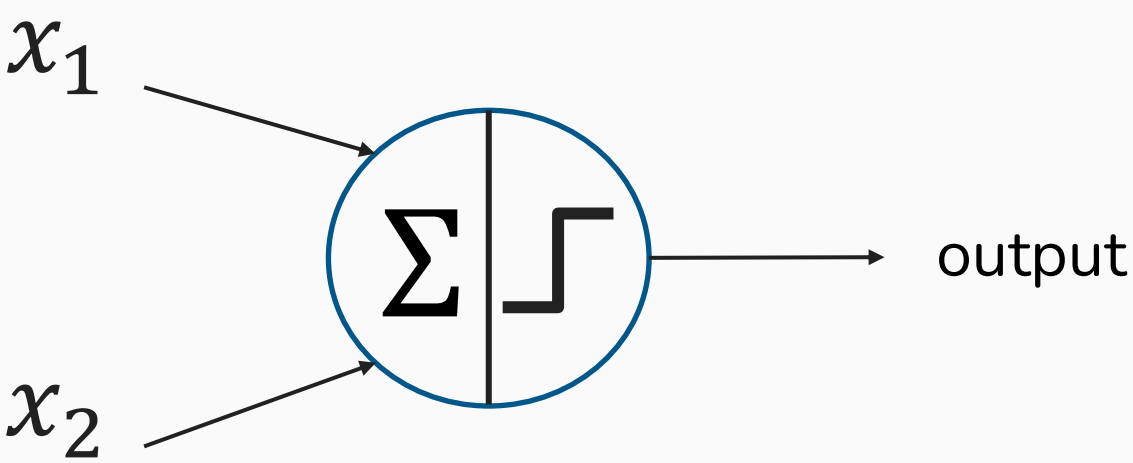
$$\begin{cases} \sum_i x_i < 2 \\ \sum_i x_i \geq 2 \end{cases}$$



With a threshold of 2.0 they modeled and problem

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

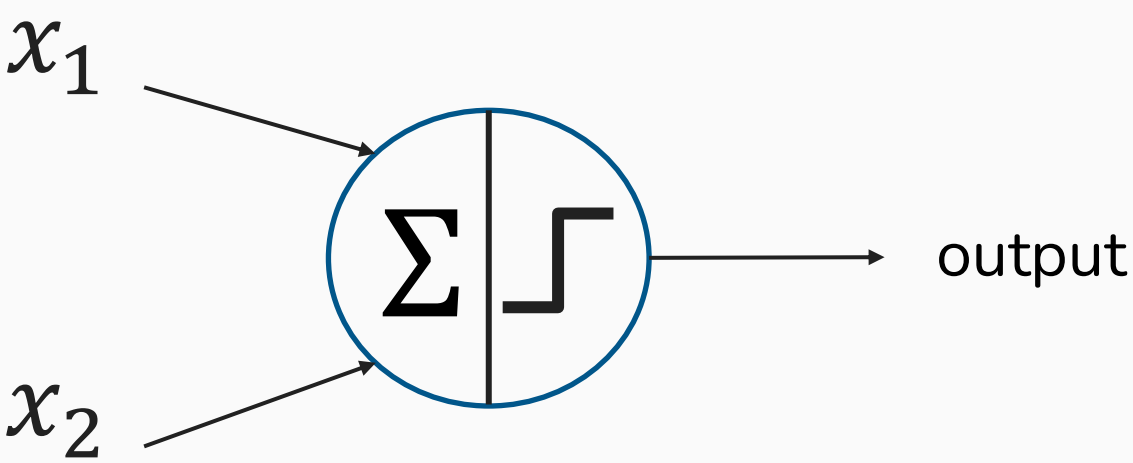
$$\begin{cases} \sum_i x_i < 2 \longrightarrow \\ \sum_i x_i \geq 2 \end{cases}$$



With a threshold of 2.0 they modeled and problem

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

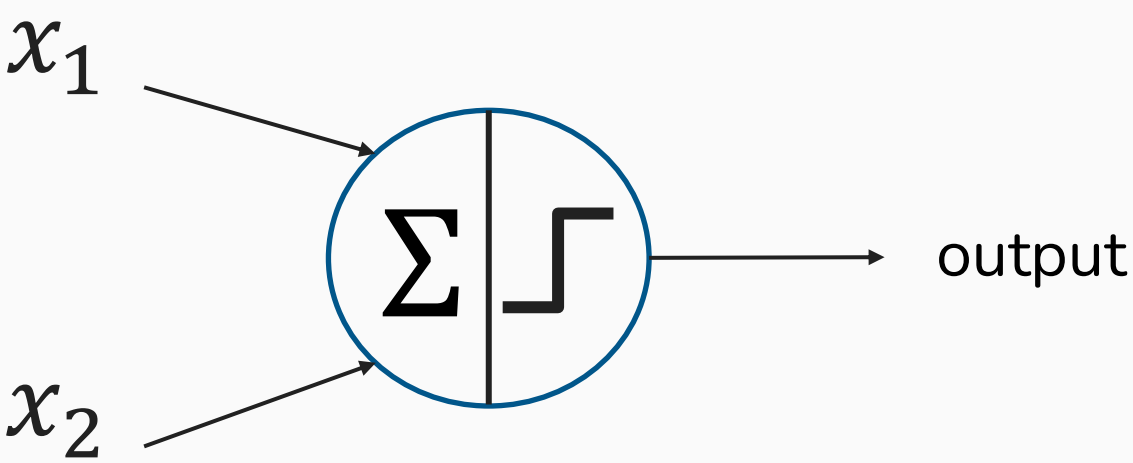
$$\begin{cases} \sum_i x_i < 2 \longrightarrow \text{Output} = 0 \\ \sum_i x_i \geq 2 \end{cases}$$



With a threshold of 2.0 they modeled and problem

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{cases} \sum_i x_i < 2 \longrightarrow \text{Output} = 0 \\ \sum_i x_i \geq 2 \longrightarrow \end{cases}$$

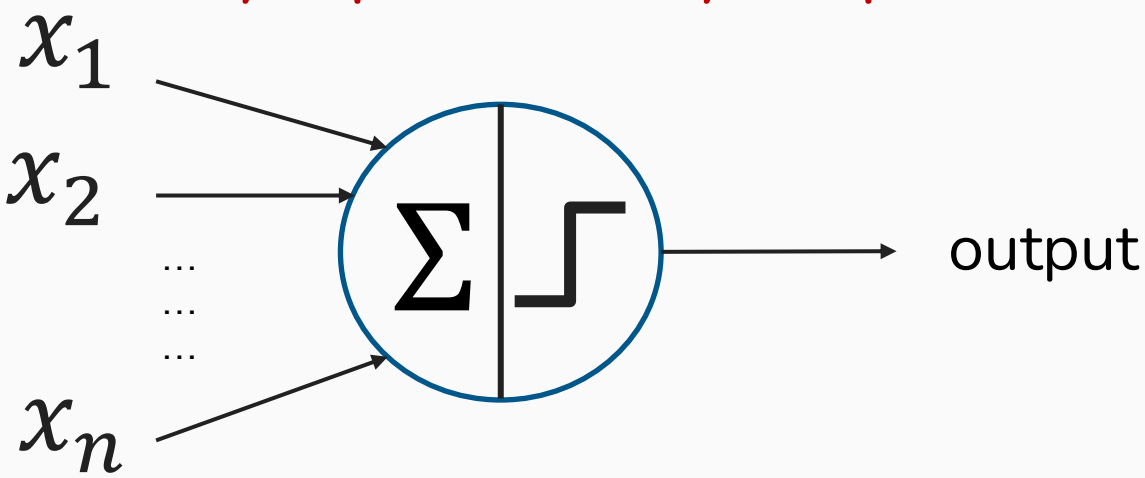


With a threshold of 2.0 they modeled and problem

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{cases} \sum_i x_i < 2 \longrightarrow \text{Output} = 0 \\ \sum_i x_i \geq 2 \longrightarrow \text{Output} = 1 \end{cases}$$

Binary inputs Binary outputs



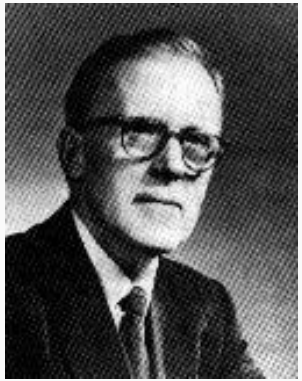
With a threshold of 2.0 they modeled and problem

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{cases} \sum_i x_i < 2 \longrightarrow \text{Output} = 0 \\ \sum_i x_i \geq 2 \longrightarrow \text{Output} = 1 \end{cases}$$

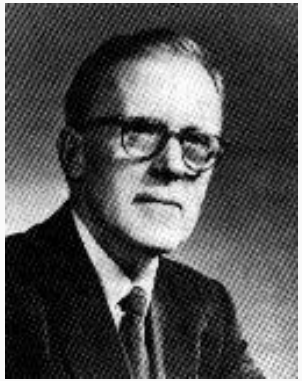
1949

1949

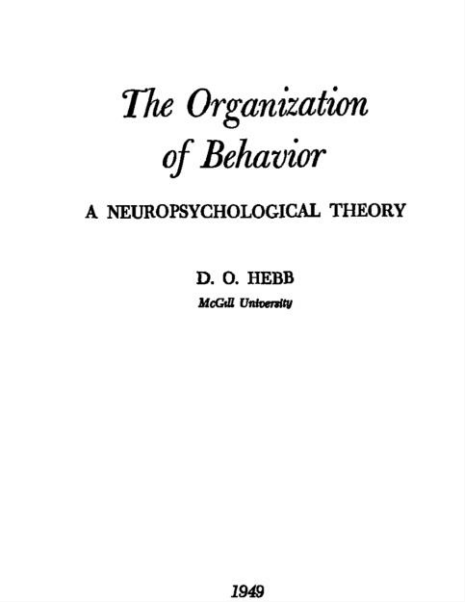


Donald Olding Hebb
1898-1969

1949



Donald Olding Hebb
1898-1969



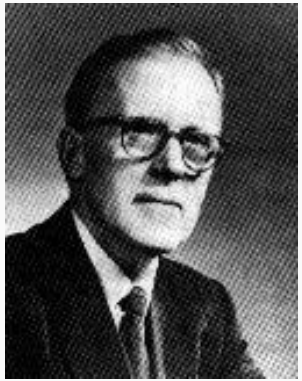
The Organization of Behavior

A NEUROPSYCHOLOGICAL THEORY

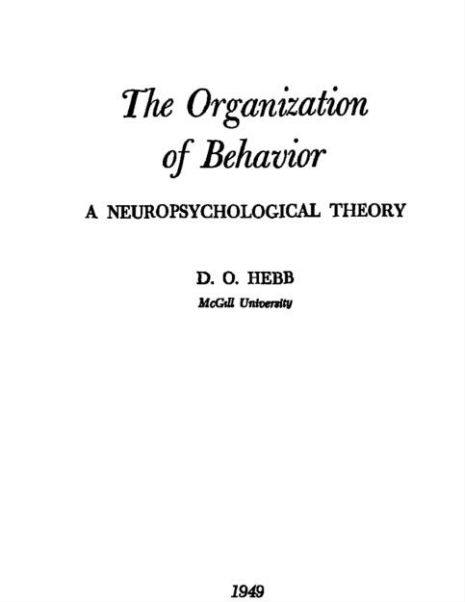
D. O. HEBB
McGill University

1949

1949



Donald Olding Hebb
1898-1969



The Organization of Behavior

A NEUROPSYCHOLOGICAL THEORY

D. O. HEBB
McGill University

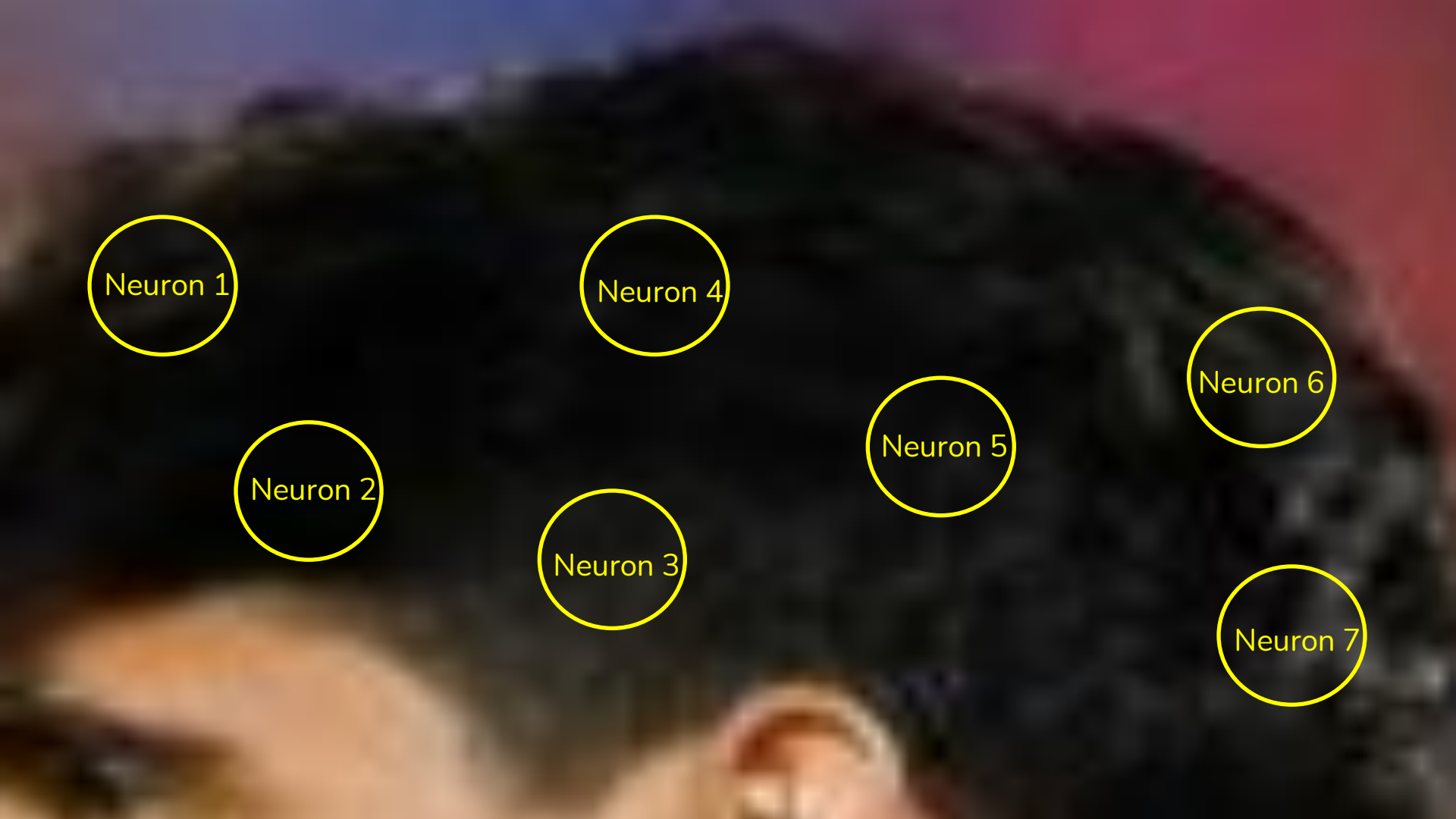
1949

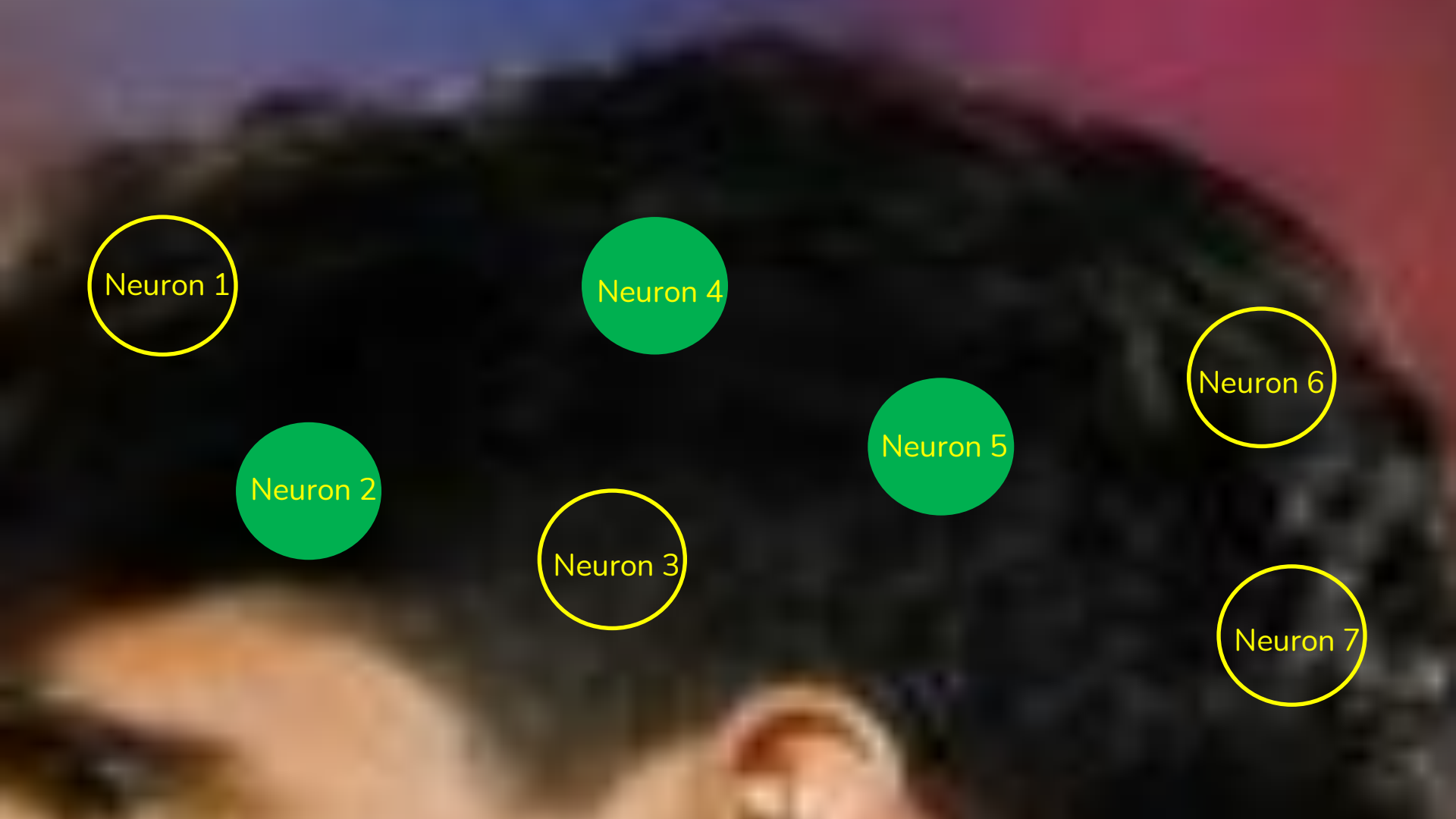
“neurons that fire together, wire together”

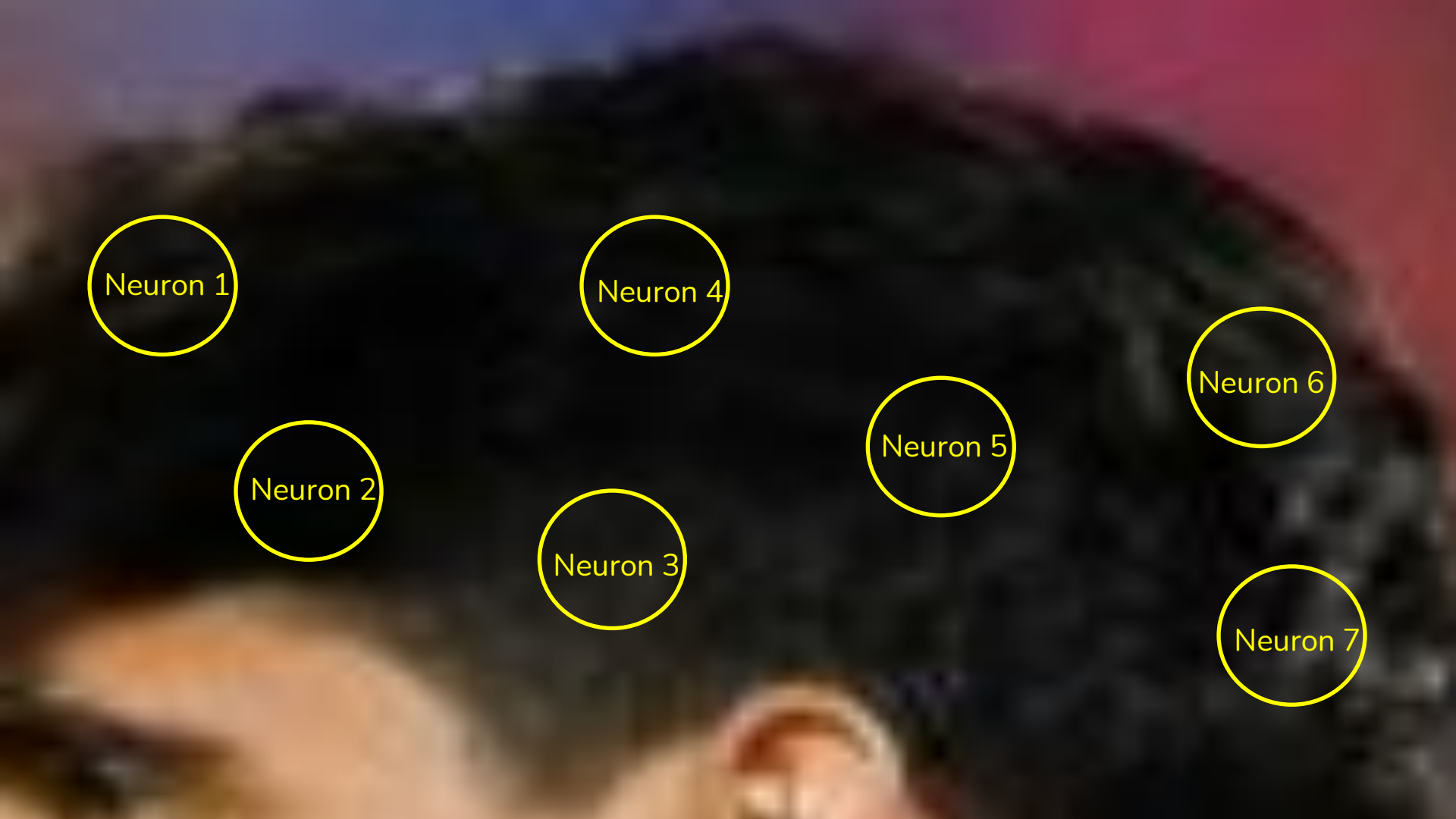
25
15
23

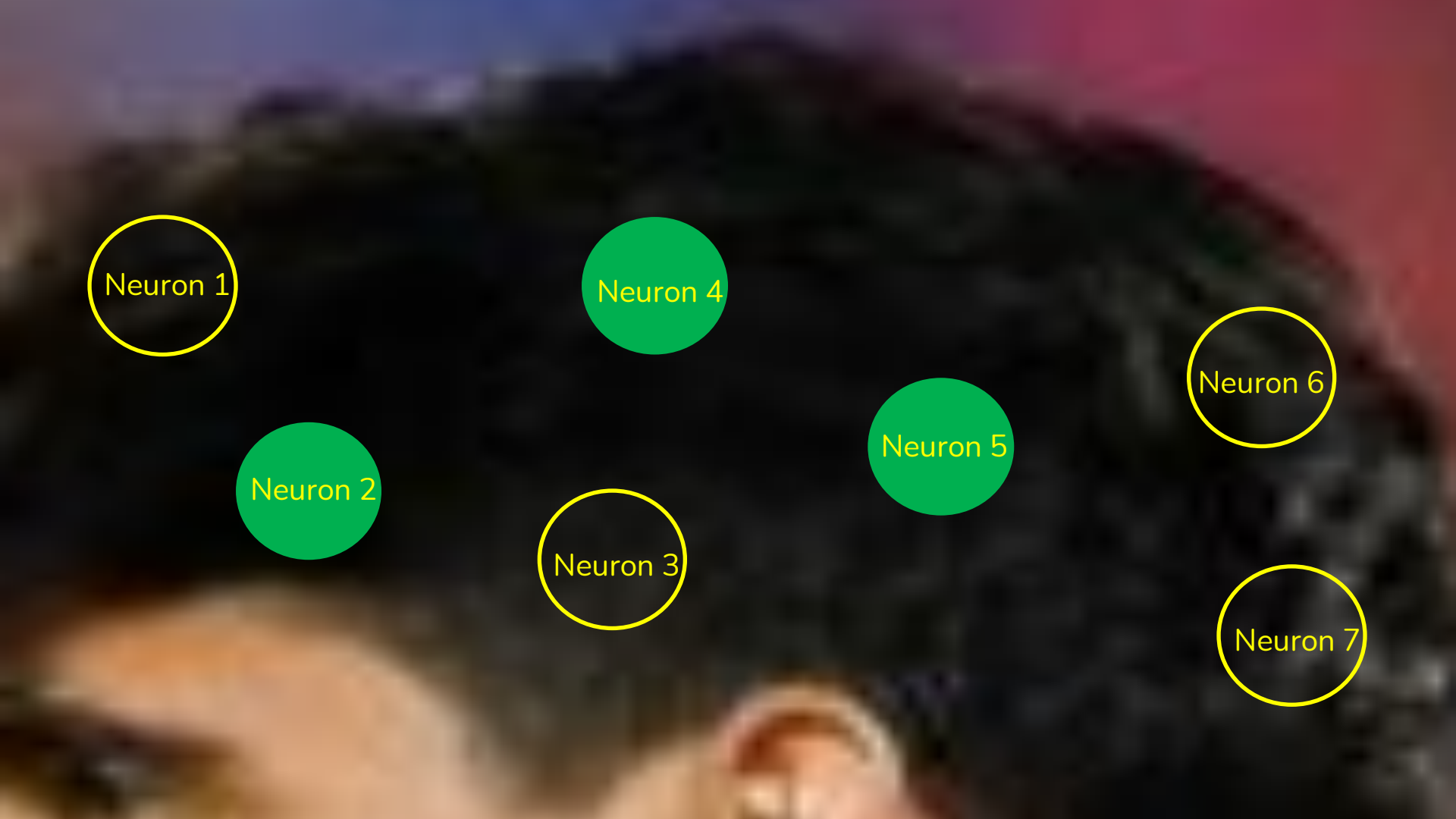
25
15
23
9

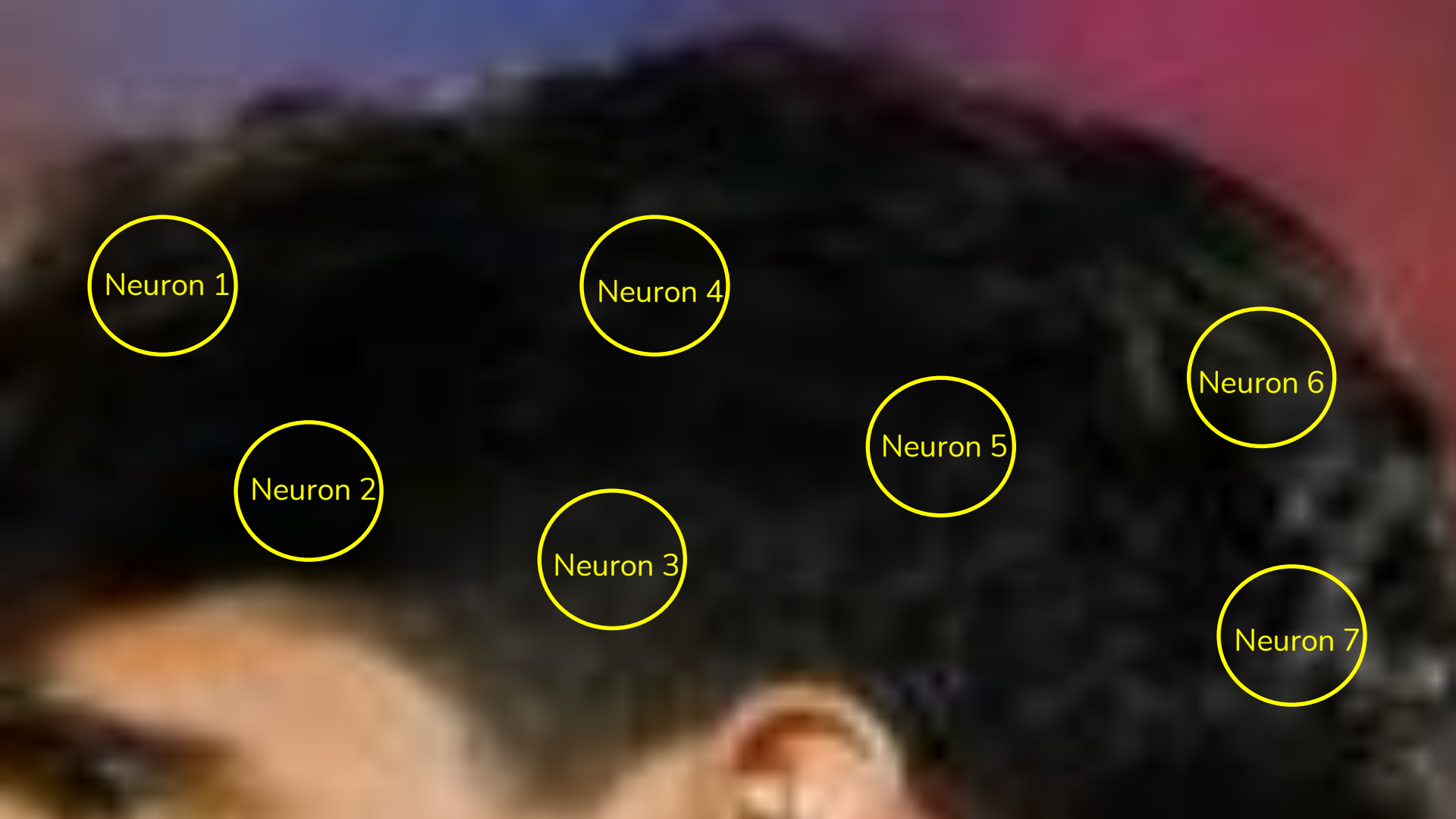
MARATHON
BET
13
everf

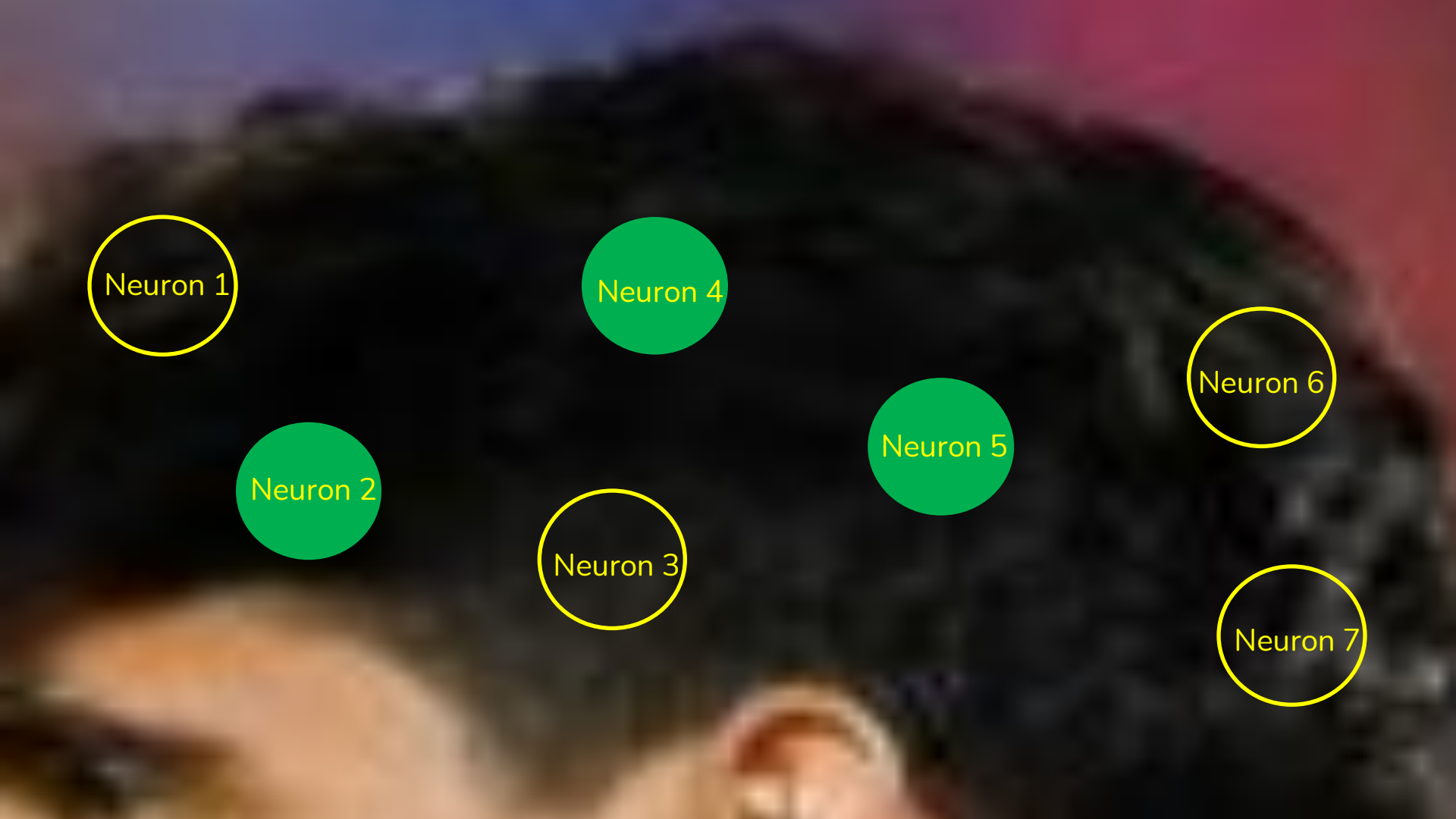

Neuron 1
Neuron 2
Neuron 3
Neuron 4
Neuron 5
Neuron 6
Neuron 7

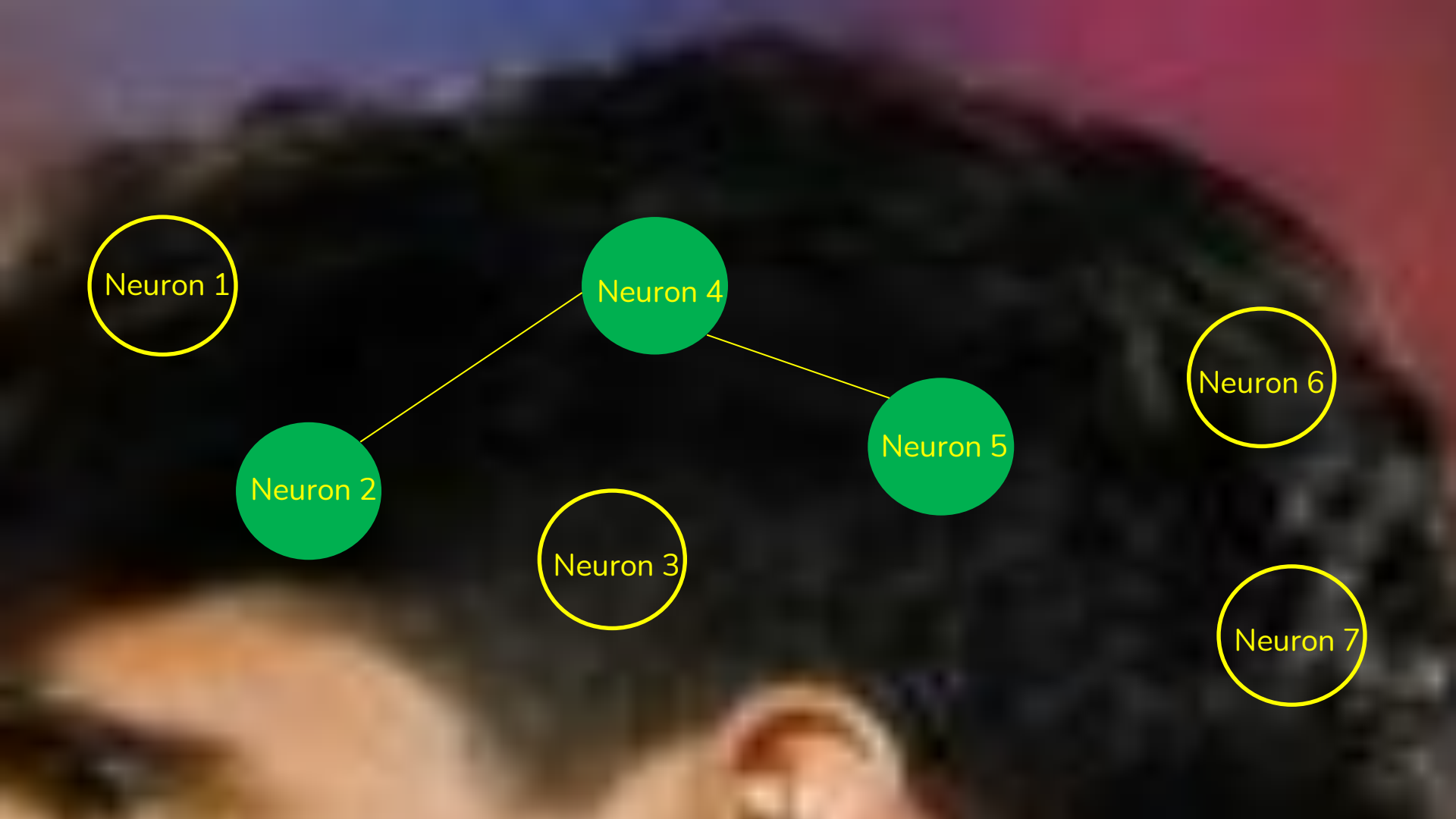

Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7


Neuron 1
Neuron 2
Neuron 3
Neuron 4
Neuron 5
Neuron 6
Neuron 7

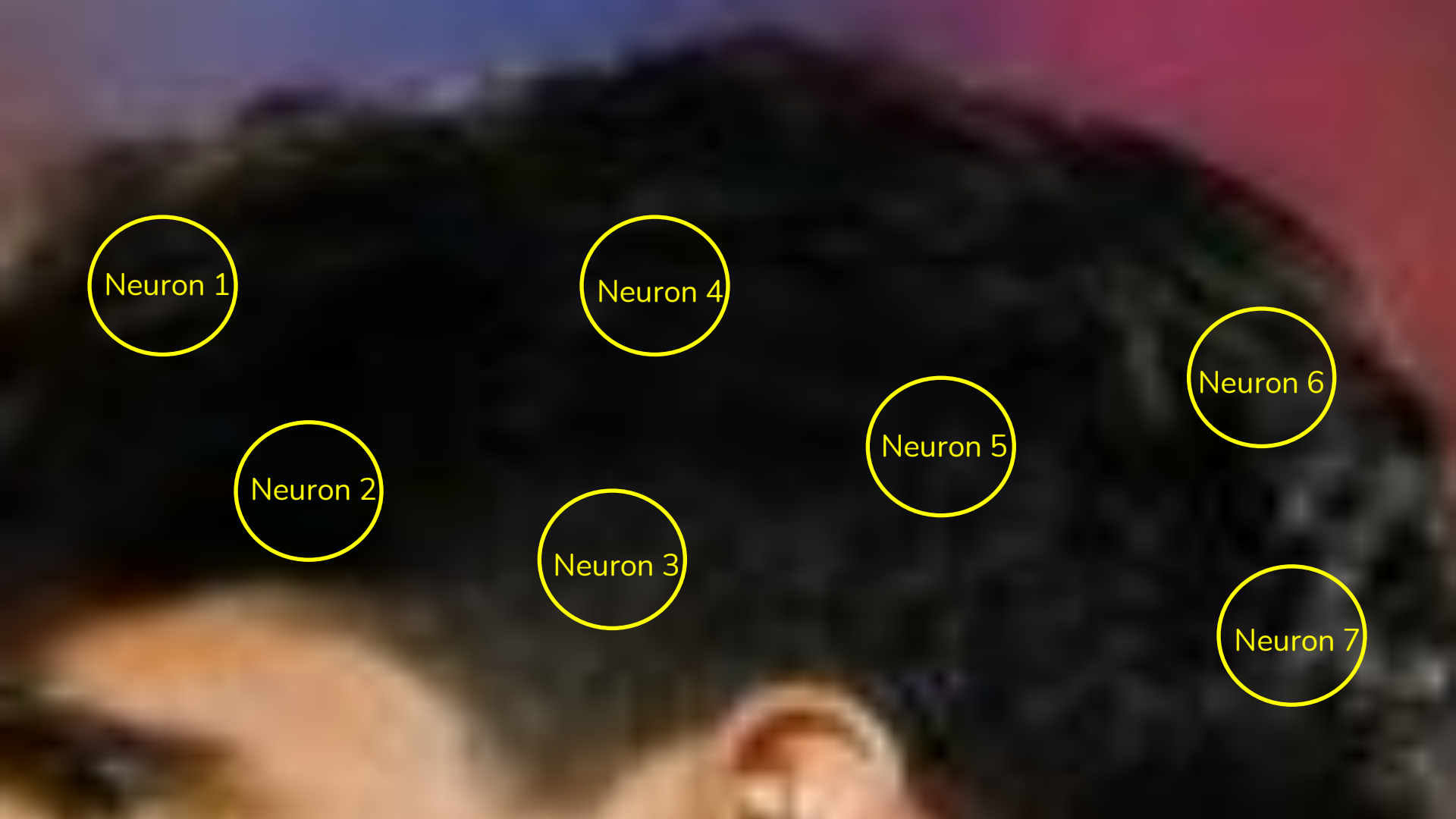

Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7

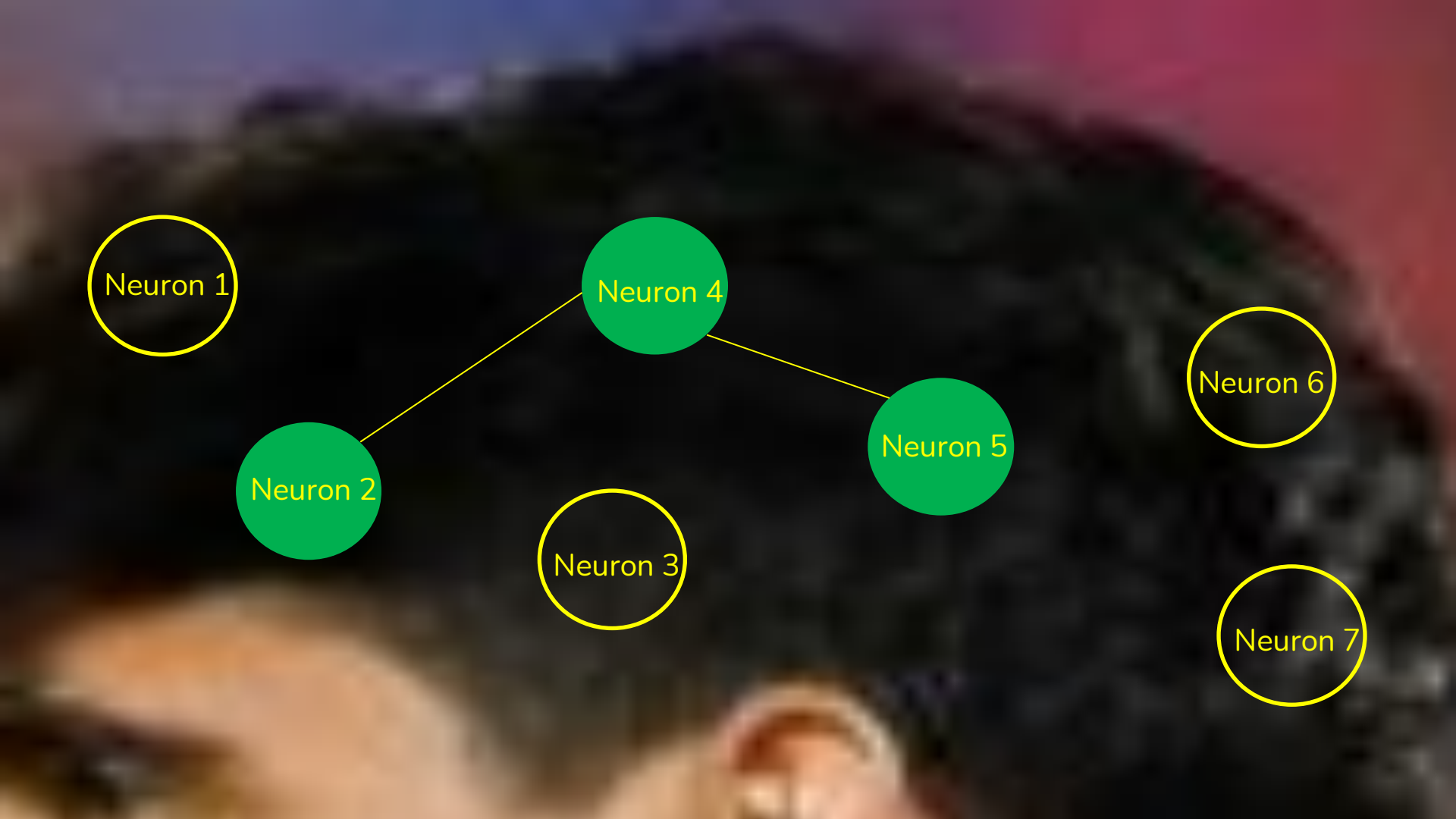

Neuron 1
Neuron 2
Neuron 3
Neuron 4
Neuron 5
Neuron 6
Neuron 7

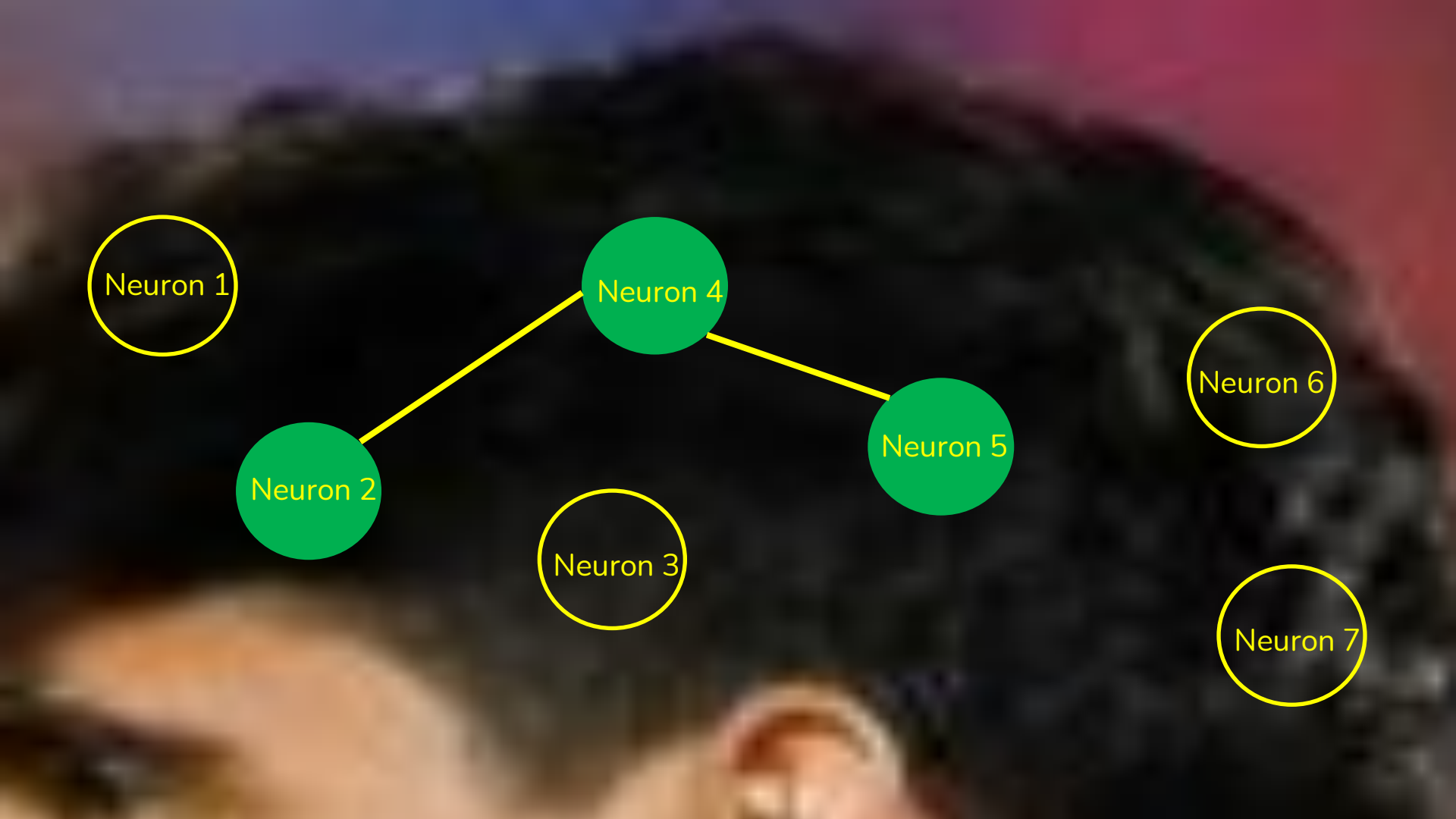

Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7

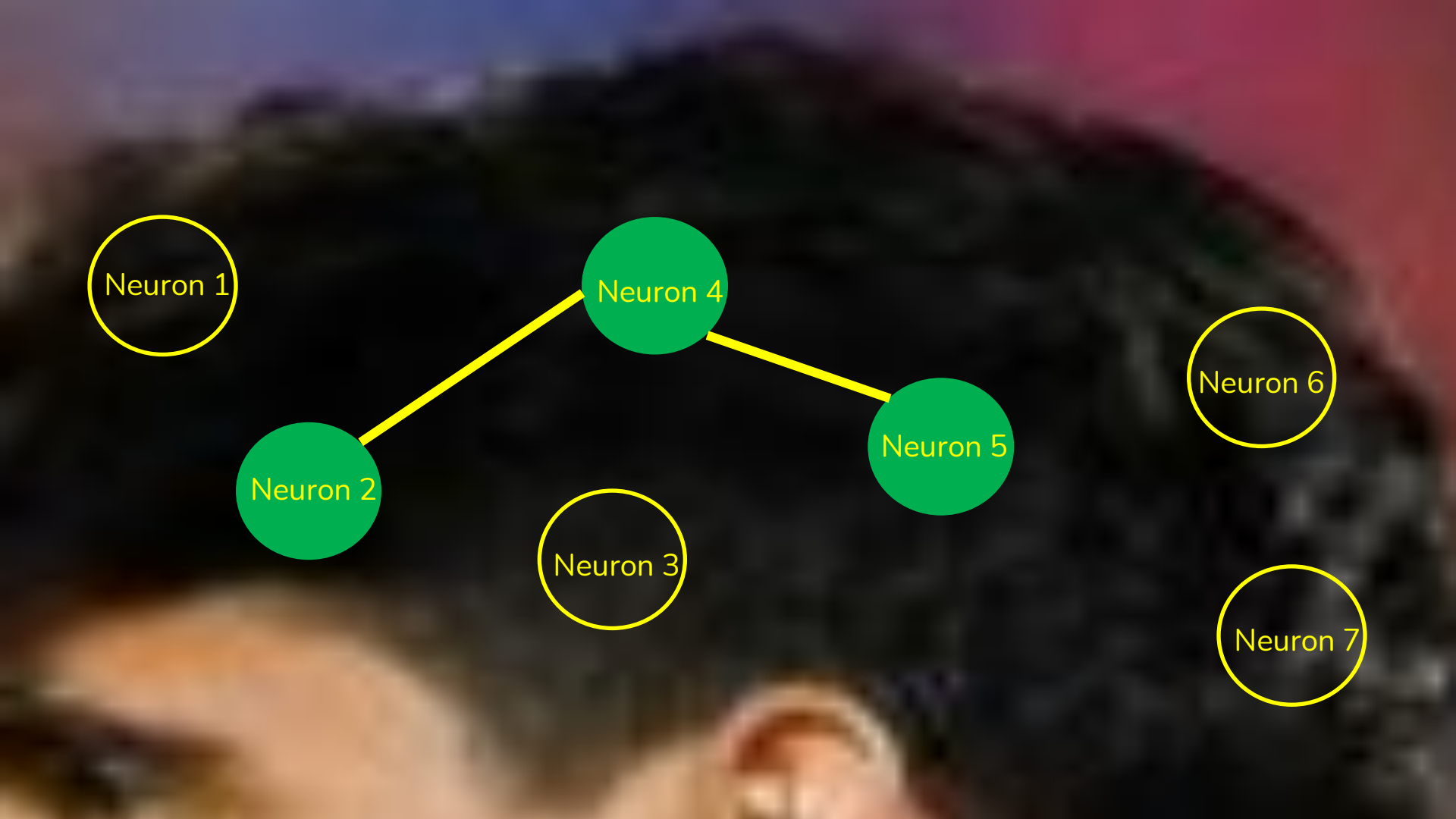

Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7

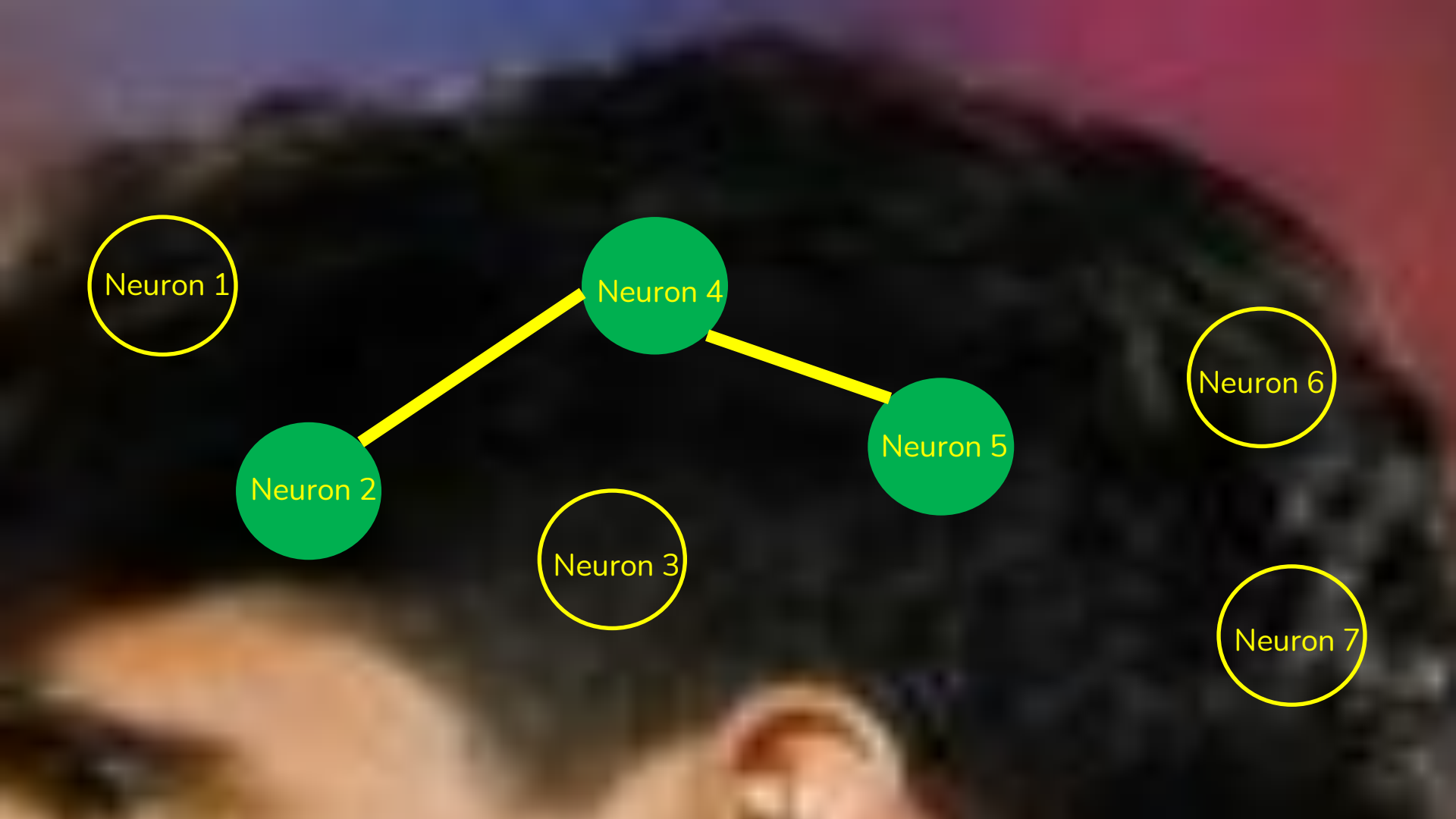
PS5
PLAY


Neuron 1
Neuron 2
Neuron 3
Neuron 4
Neuron 5
Neuron 6
Neuron 7


Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7


Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7


Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7

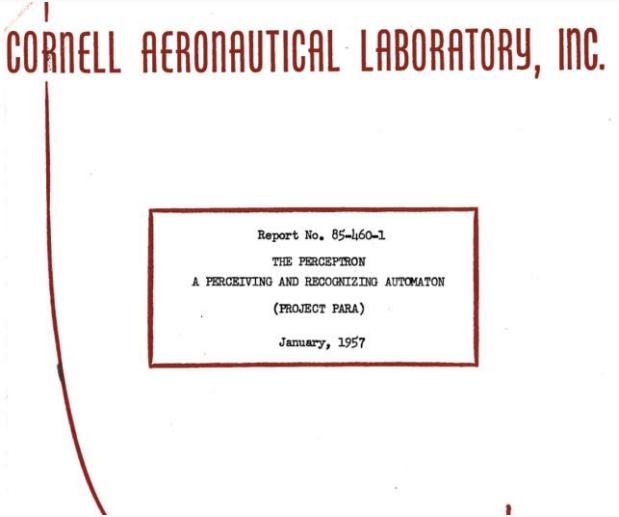

Neuron 1
Neuron 4
Neuron 6
Neuron 5
Neuron 2
Neuron 3
Neuron 7

1957

Frank Rosenblatt
1928-1971

1957

Frank Rosenblatt
1928-1971



CORNELL AERONAUTICAL LABORATORY, INC.

Report No. 85-460-1

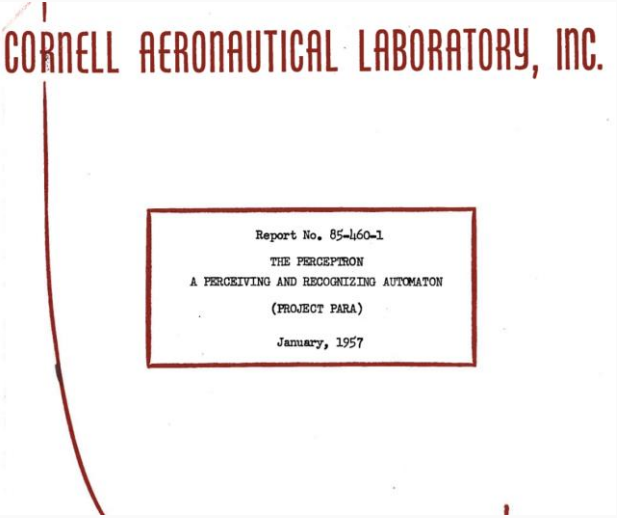
THE PERCEPTRON
A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

1957

Frank Rosenblatt
1928-1971



CORNELL AERONAUTICAL LABORATORY, INC.

Report No. 85-460-1

THE PERCEPTRON
A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

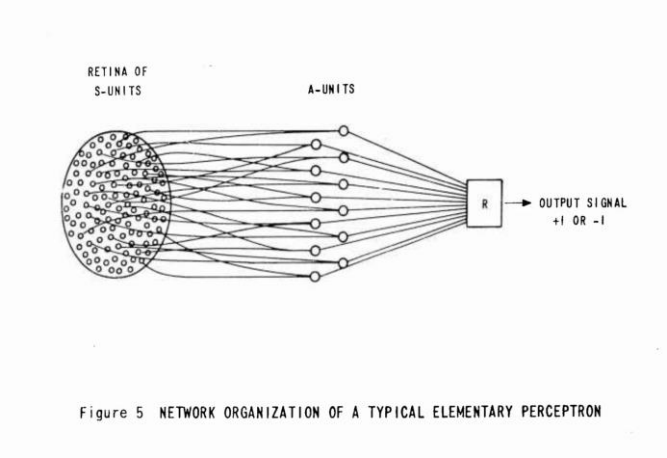
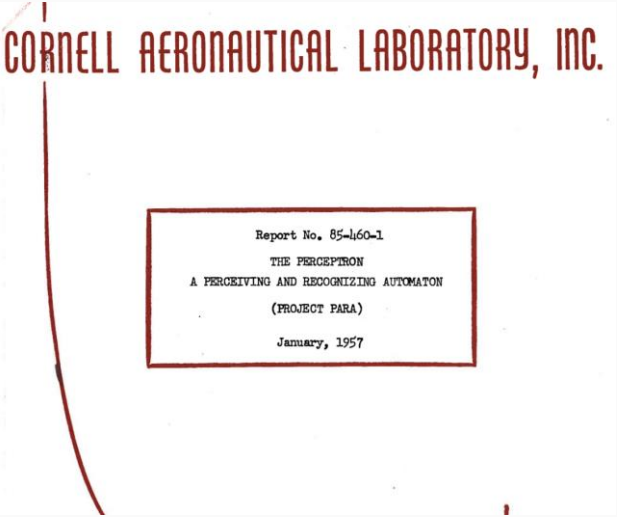


Figure 5 NETWORK ORGANIZATION OF A TYPICAL ELEMENTARY PERCEPTRON

1957

Frank Rosenblatt
1928-1971



CORNELL AERONAUTICAL LABORATORY, INC.

Report No. 85-460-1

THE PERCEPTRON
A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

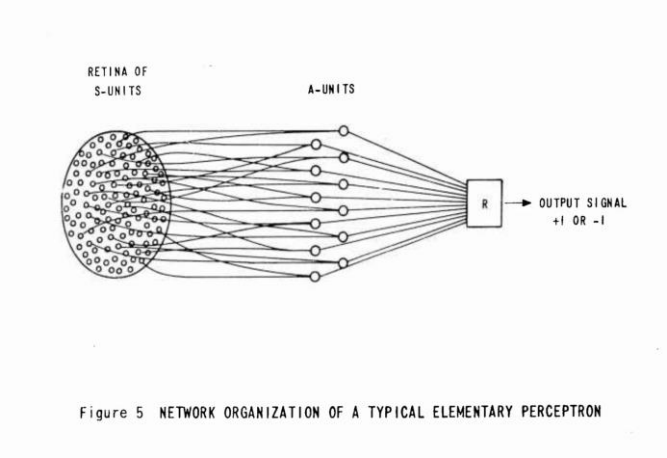
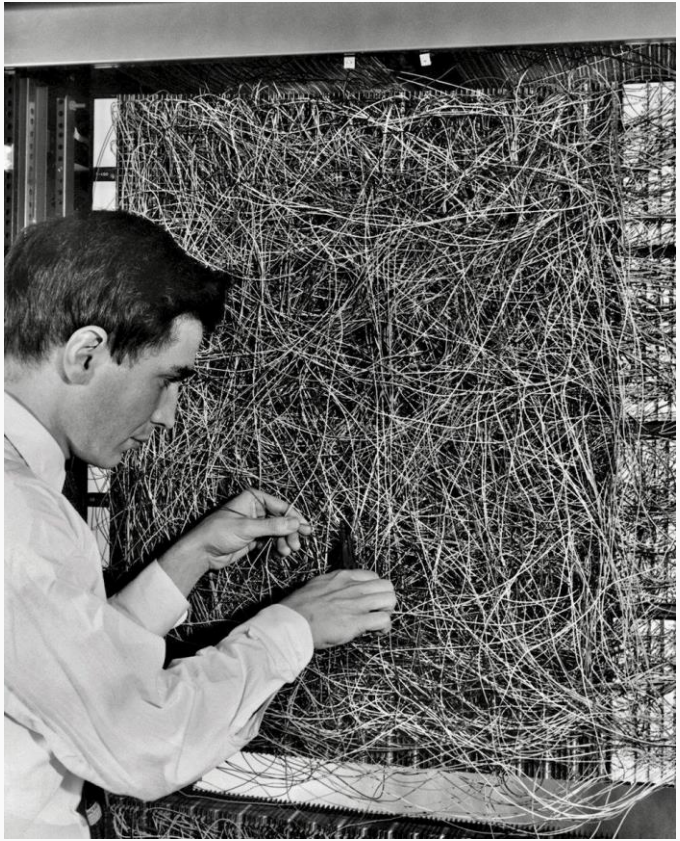


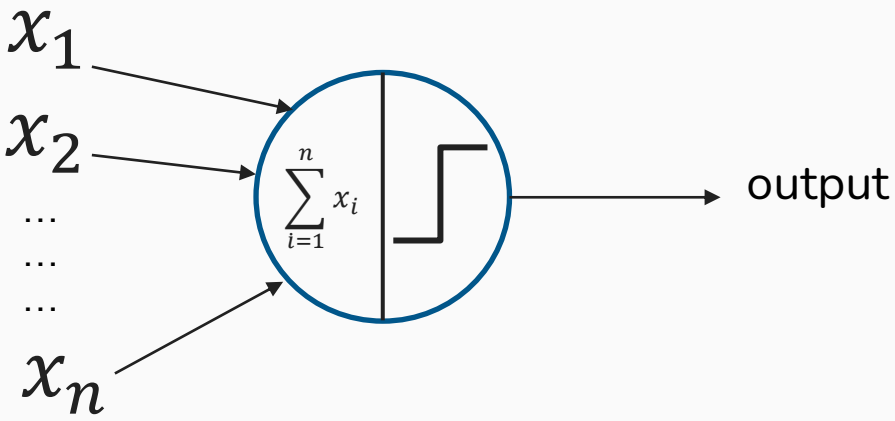
Figure 5 NETWORK ORGANIZATION OF A TYPICAL ELEMENTARY PERCEPTRON

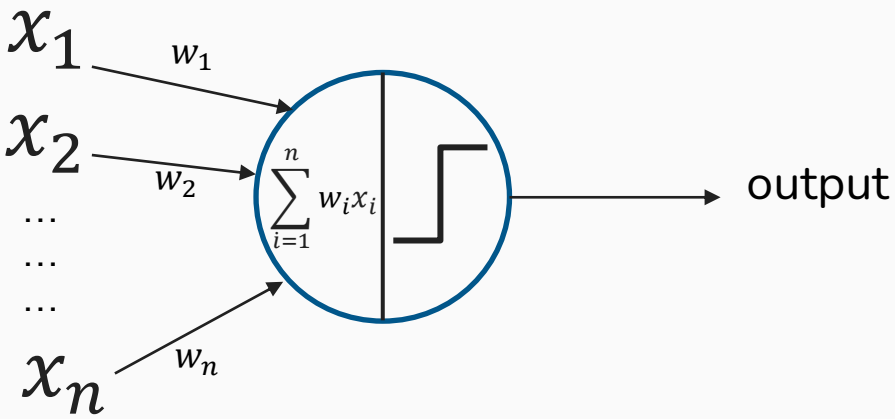
« the perceptron»

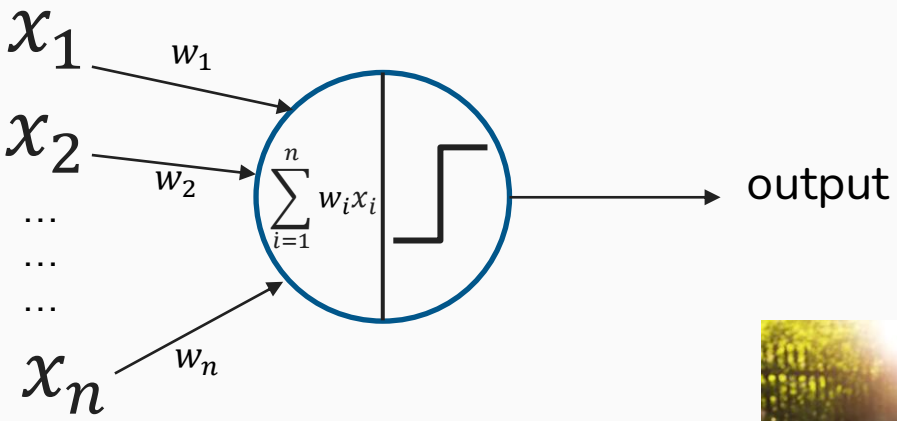
Mark 1 perceptron

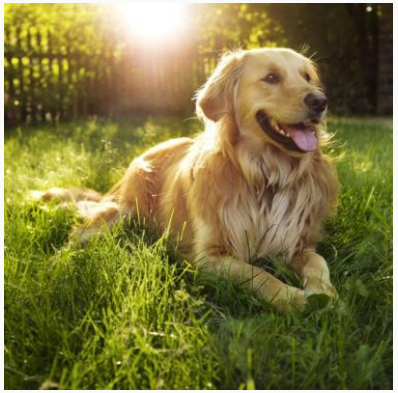


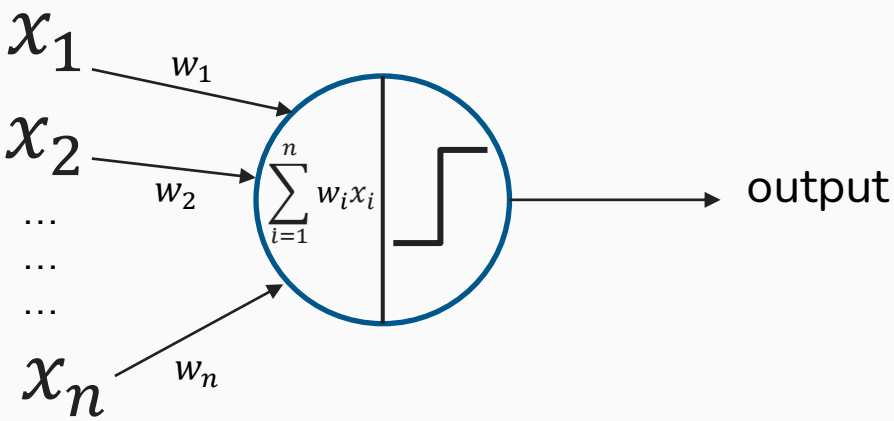
Earlier the term perceptron referred to a single layer neural net but nowadays it reffers to a single neuron :

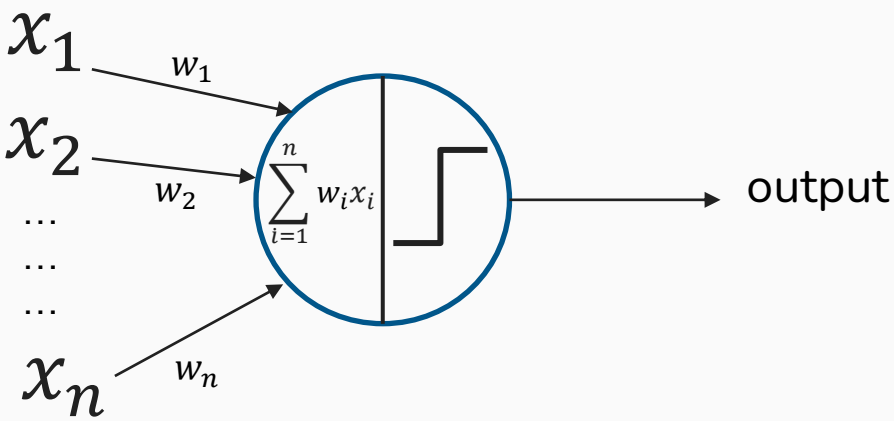

x_1
x_2
…
…
…
x_n
$\sum_{i=1}^{n} x_i$
output


x_1
x_2
…
…
…
x_n
w_1
w_2
w_n
$\sum_{i=1}^{n} w_i x_i$
output

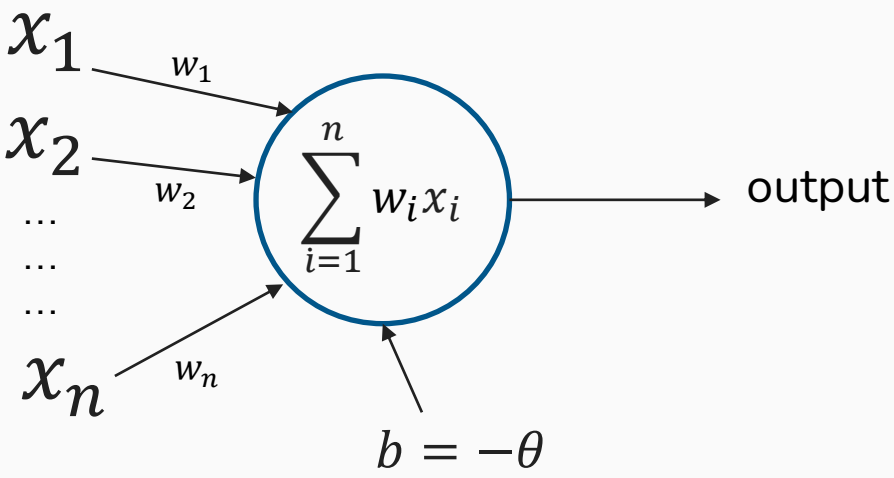

x_1
w_1
x_2
w_2
…
…
…
x_n
w_n
$\sum_{i=1}^{n} w_i x_i$
output




x_1
x_2
…
…
…
x_n
w_1
w_2
w_n
$\sum_{i=1}^{n} w_i x_i$
output

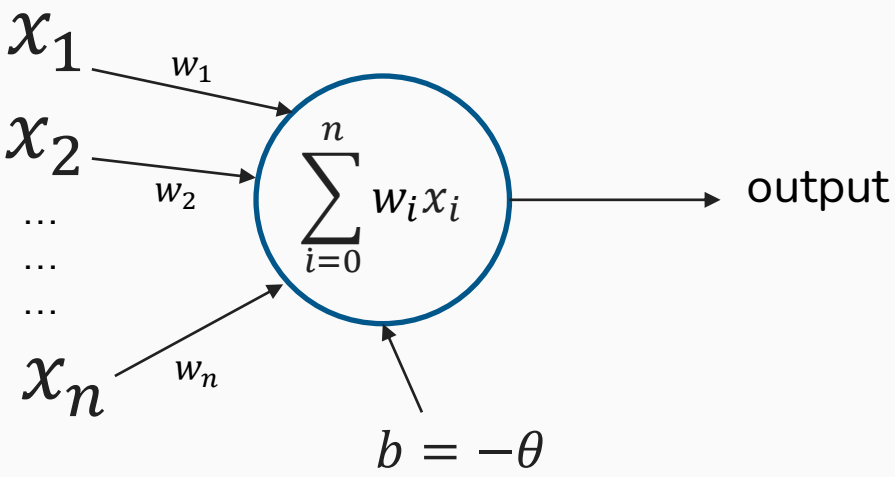


$$\begin{cases} \sum_{i=1}^{n} w_i x_i < \theta \longrightarrow \text{Output} = 0 \\ \sum_{i=1}^{n} w_i x_i \geq \theta \longrightarrow \text{Output} = 1 \end{cases}$$



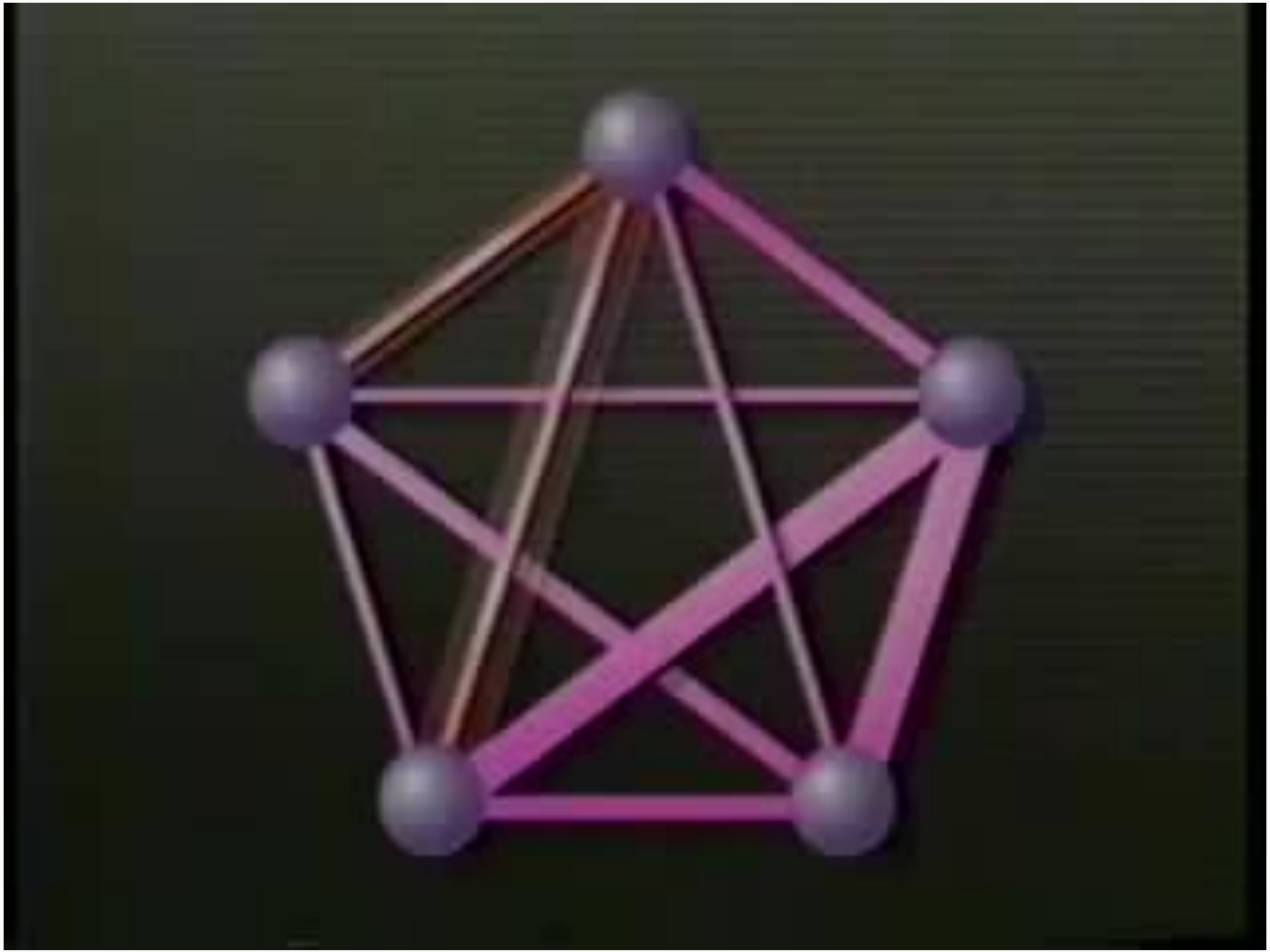
$$x_0 = 1$$

$$\begin{cases} \sum_{i=1}^{n} w_i x_i < \theta \longrightarrow \text{Output} = 0 \\ \sum_{i=1}^{n} w_i x_i \geq \theta \longrightarrow \text{Output} = 1 \end{cases}$$



$x_0 = 1$

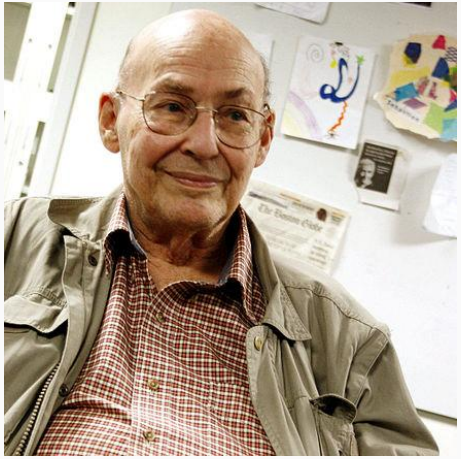
$$\begin{cases} \sum_{i=0}^{n} w_i x_i < 0 \longrightarrow \text{Output} = 0 \\ \sum_{i=0}^{n} w_i x_i \geq 0 \longrightarrow \text{Output} = 1 \end{cases}$$



Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.
- The New York Times in 1958 after a press conference with Rosenblatt

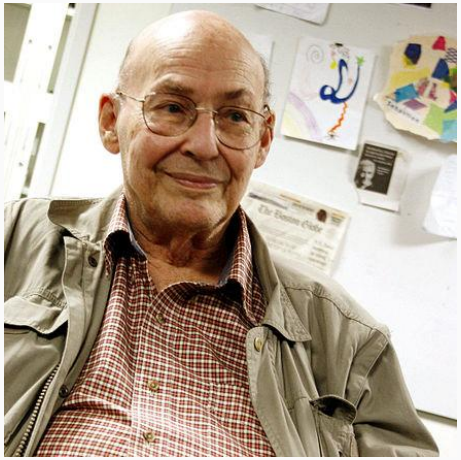
1969



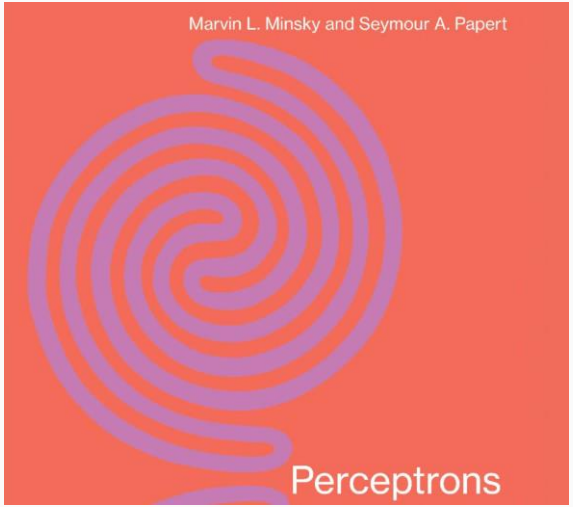
Marvin Lee Minsky
1927-2016

Seymour Aubrey Papert
1928-2016

1969

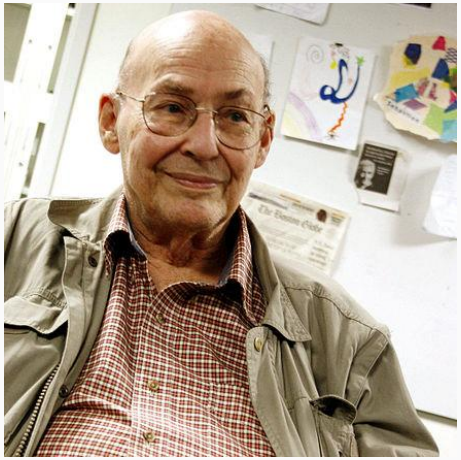


Marvin Lee Minsky
1927-2016

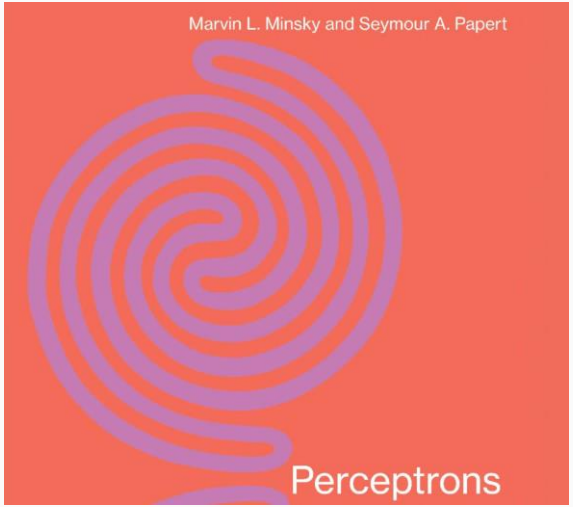


Seymour Aubrey Papert
1928-2016

1969



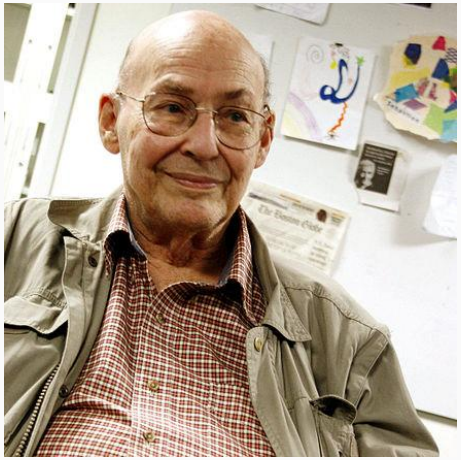
Marvin Lee Minsky
1927-2016



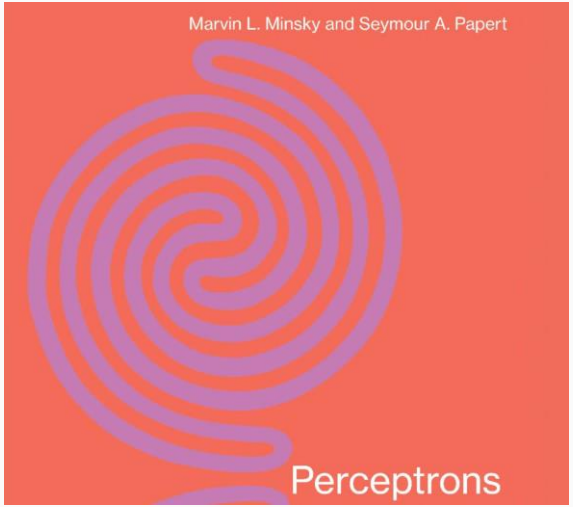
Seymour Aubrey Papert
1928-2016

Perceptrons: An Introduction to Computational Geometry

1969



Marvin Lee Minsky
1927-2016



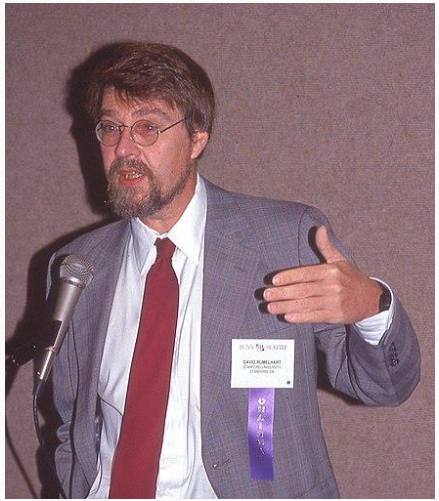
Seymour Aubrey Papert
1928-2016

Perceptrons: An Introduction to Computational Geometry

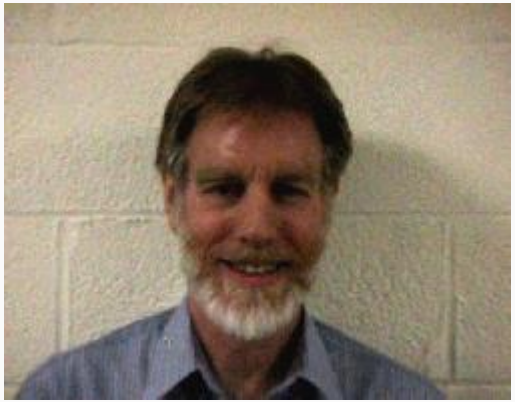
The XOR affair

THE FIRST AI WINTER

1986



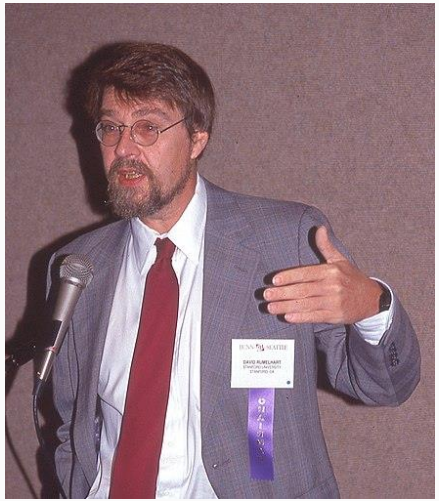
David Everett Rumelhart
1942-2011



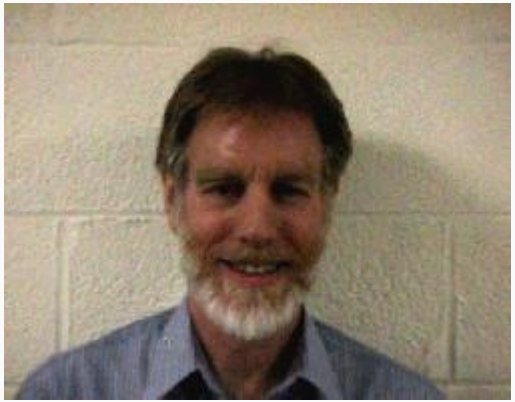
Ronald J. Williams
????-NOW

Geoffrey Everest Hinton
1947-NOW

1986



David Everett Rumelhart
1942-2011



Ronald J. Williams
????-NOW

Geoffrey Everest Hinton
1947-NOW

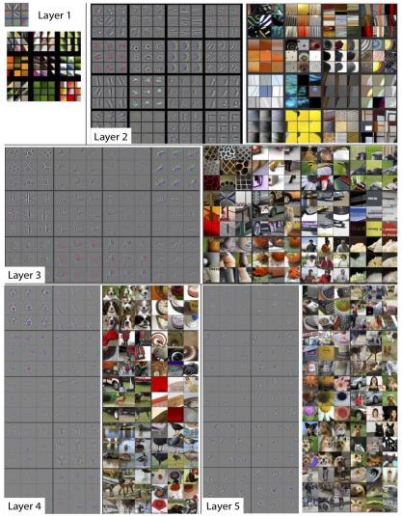
Applying BACKPROPAGATION to NN

1989

Yann LeCun
1960-NOW

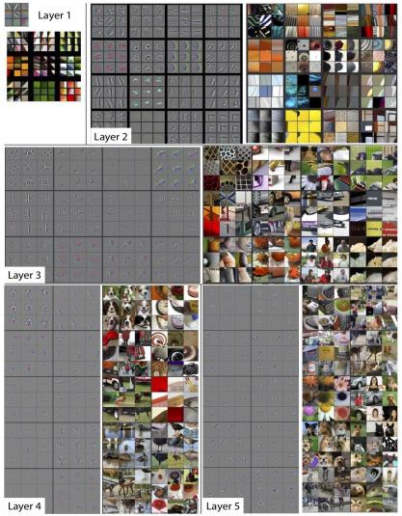
1989

Yann LeCun
1960-NOW



1989

Yann LeCun
1960-NOW



Proposing ConvNets

THE SECOND AI WINTER

1995

Sepp Hochreiter
1967-NOW

Jürgen Schmidhuber
1963-NOW

1995

Sepp Hochreiter
1967-NOW

Jürgen Schmidhuber
1963-NOW

LSTM

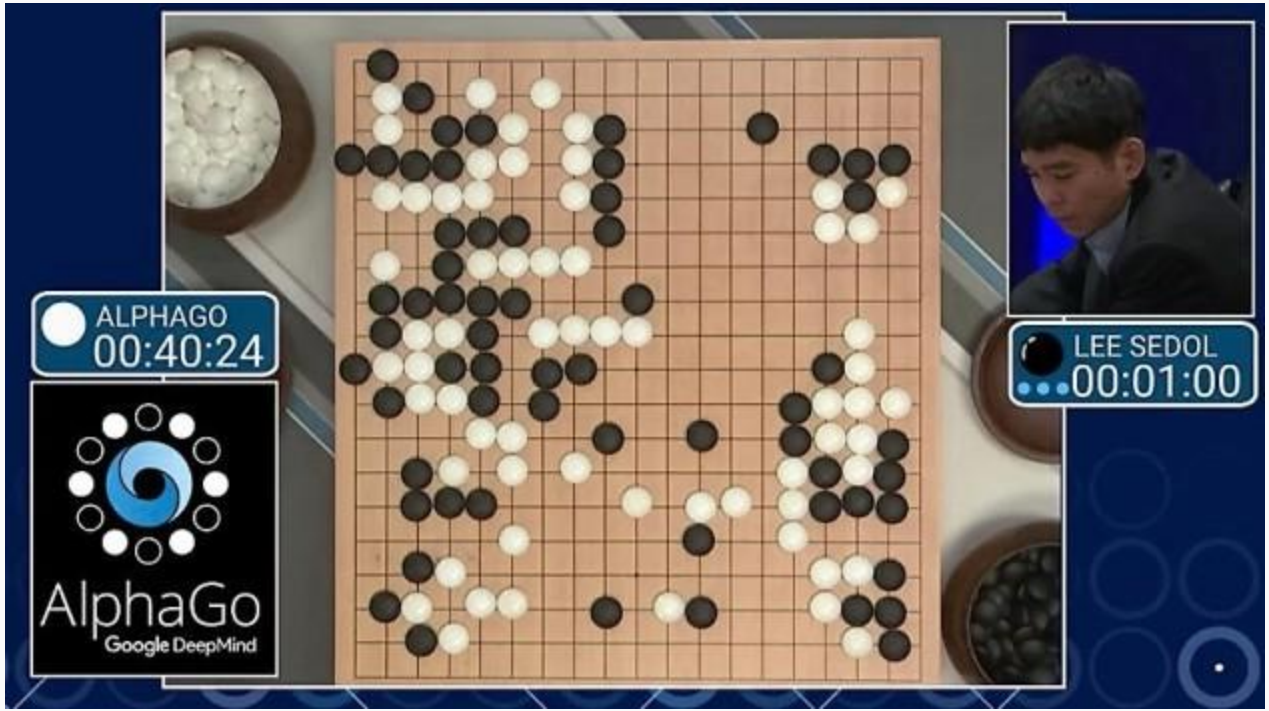
1997

1997

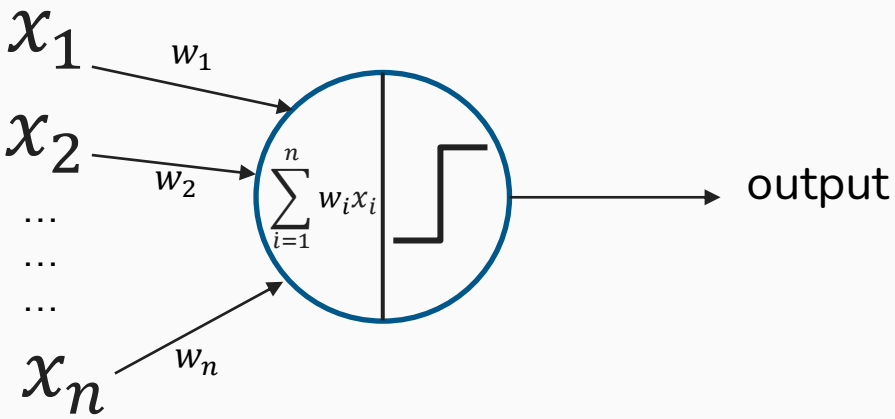
It was the first computer to win both a chess game and a chess match against a reigning world champion under regular time controls.

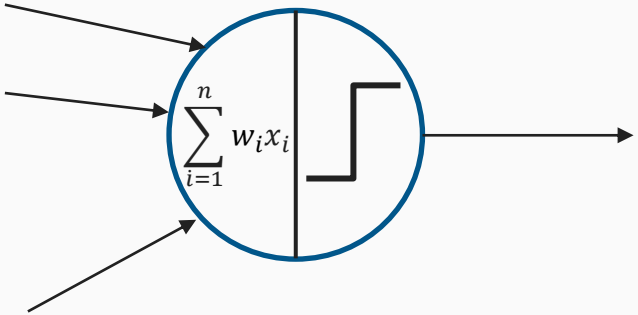
2010 : DEEP LEARNING

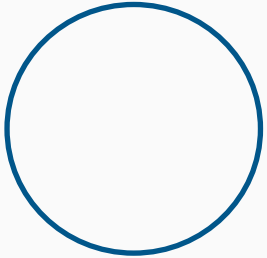
2015

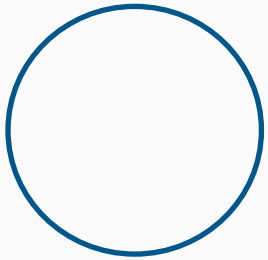


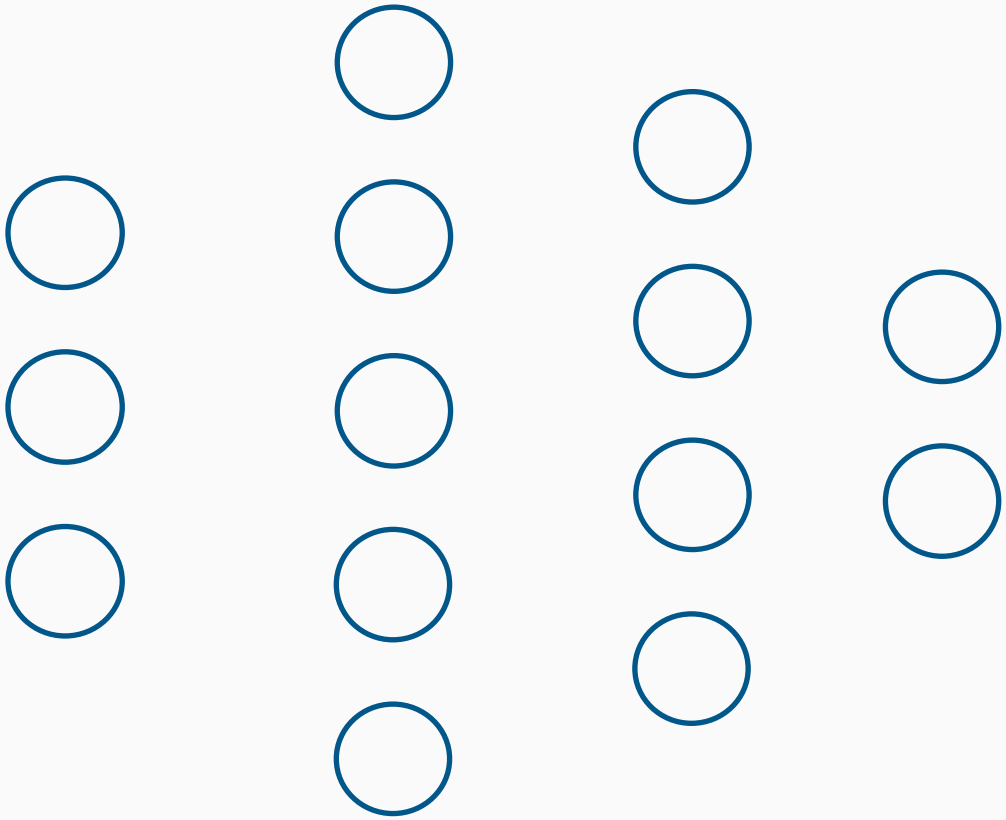
ALPHAGO BEATS A WORLD CHAMPION

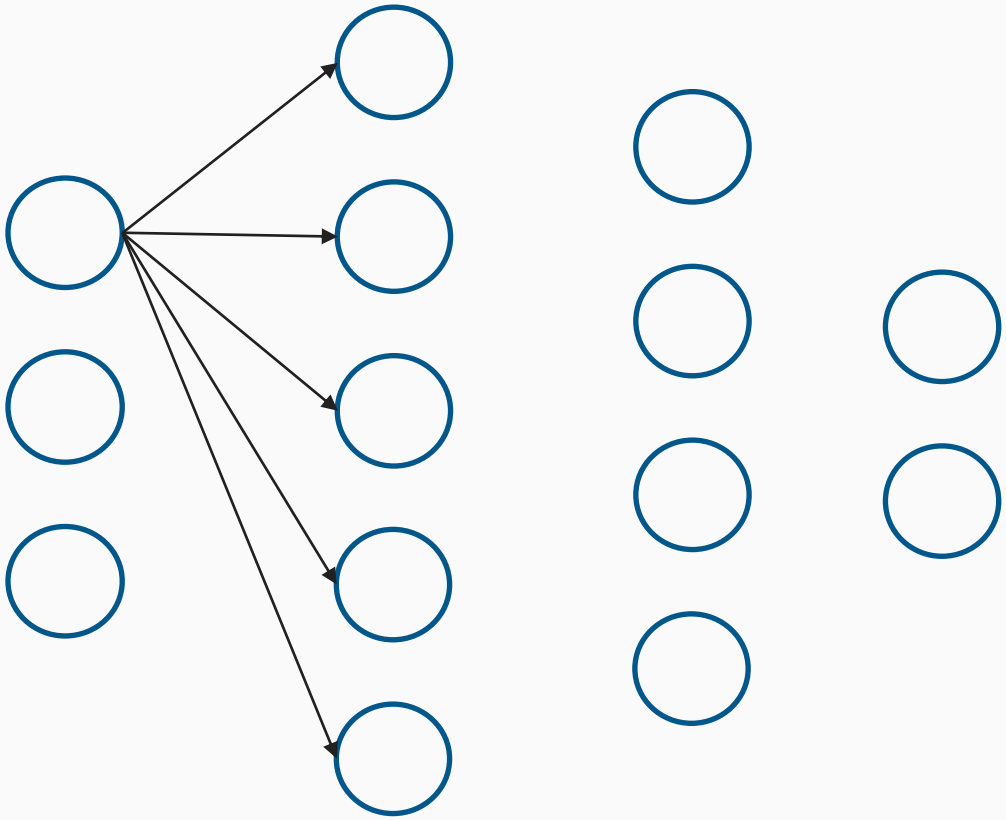

x_1
x_2
…
…
…
x_n
w_1
w_2
w_n
$\sum_{i=1}^{n} w_i x_i$
output

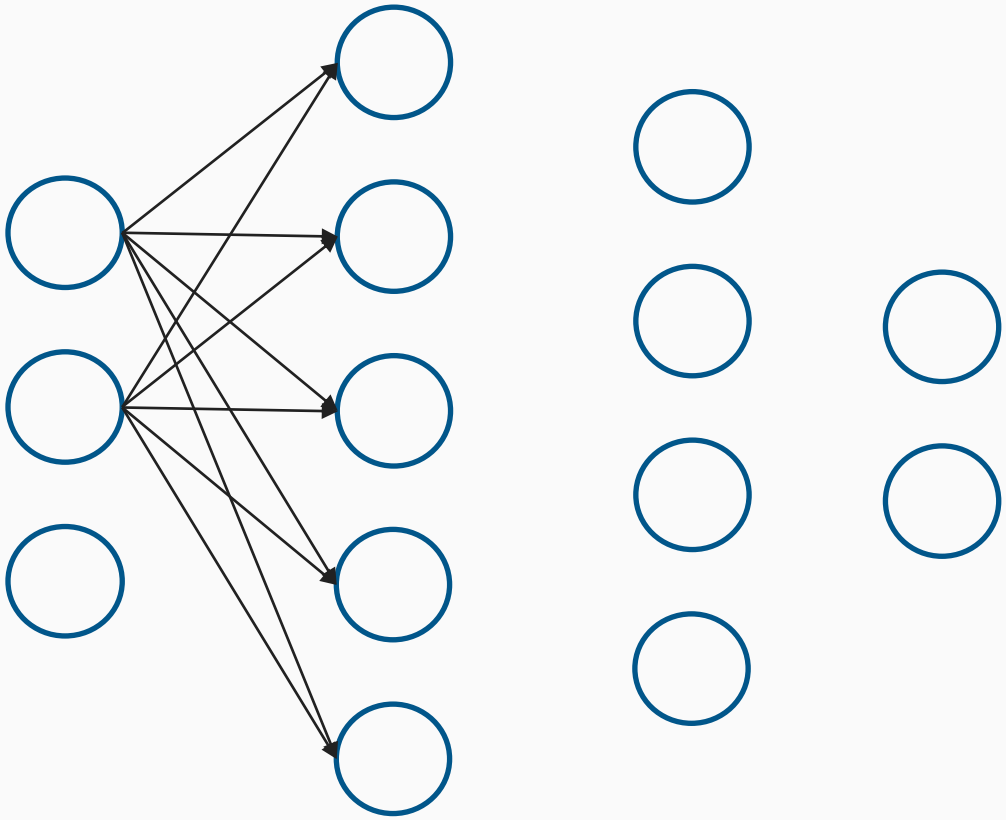

$\sum_{i=1}^{n} w_i x_i$

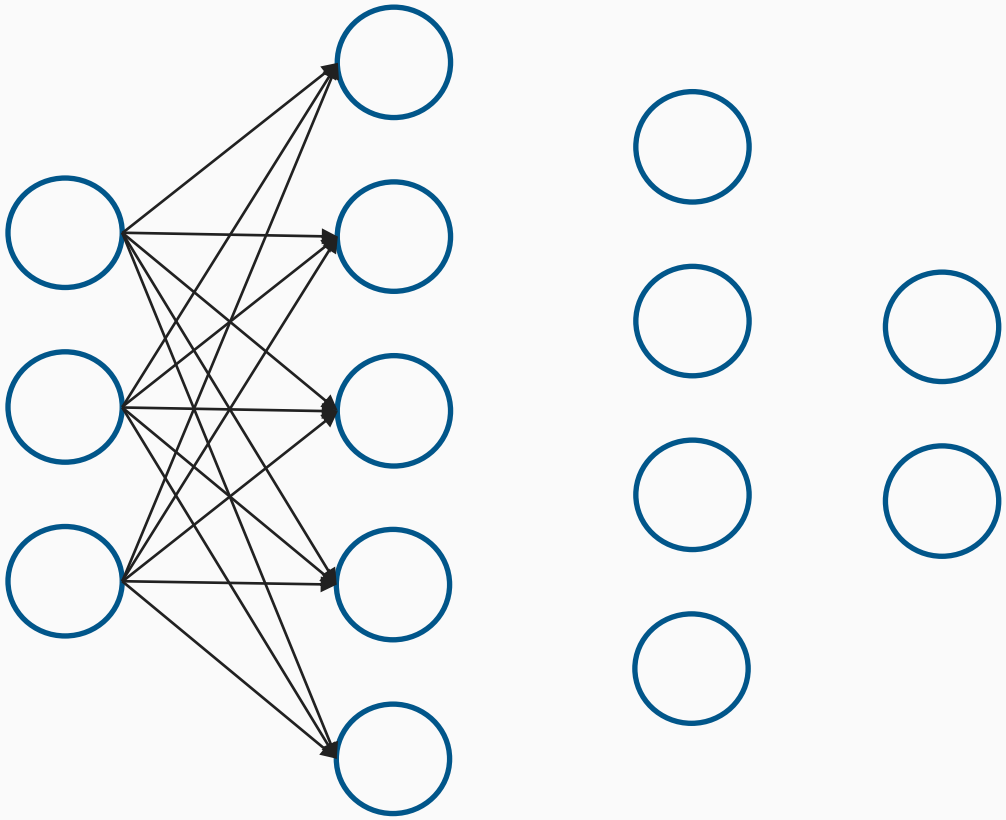


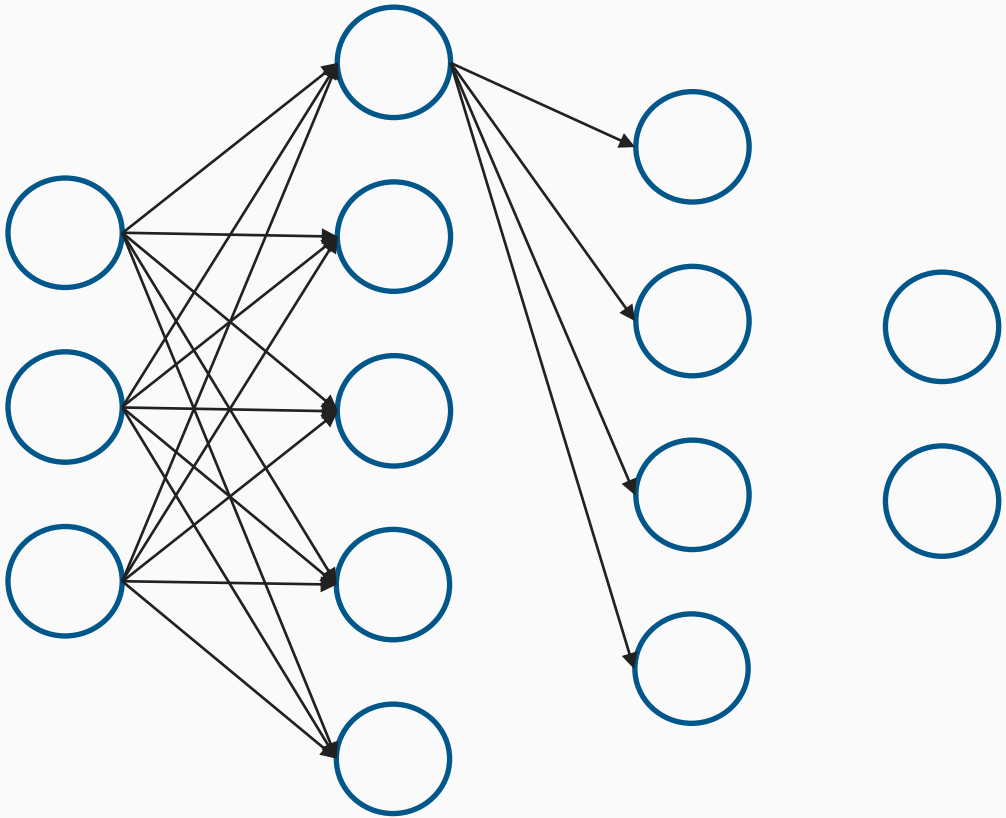


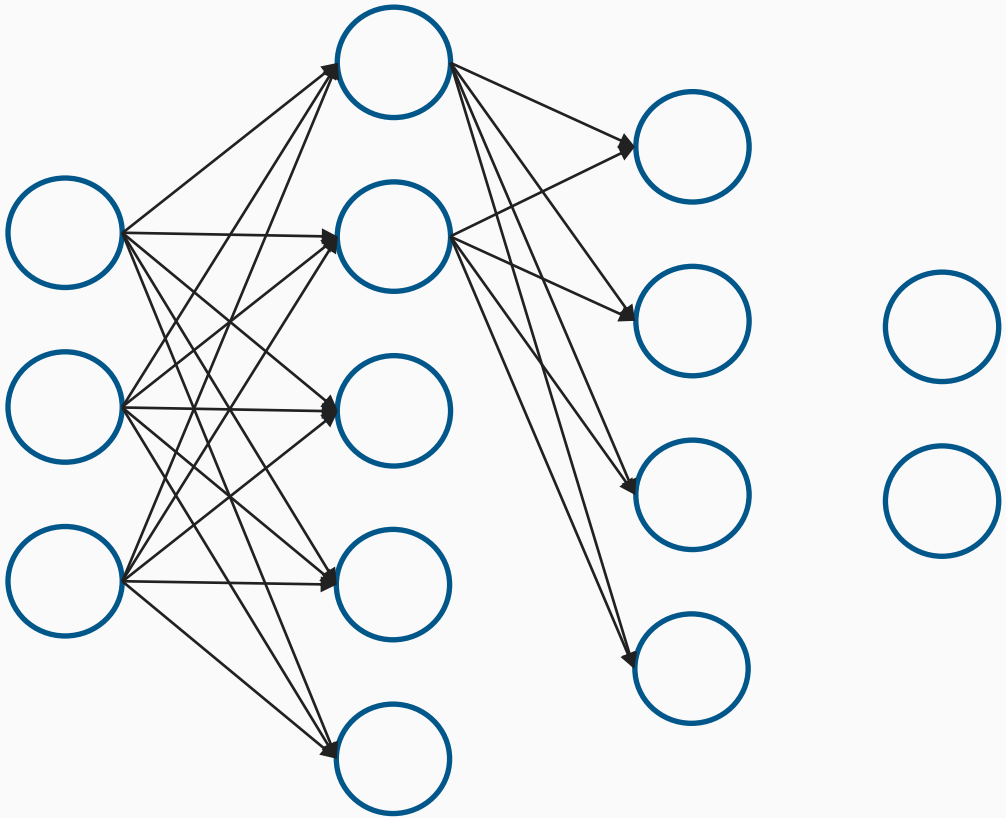


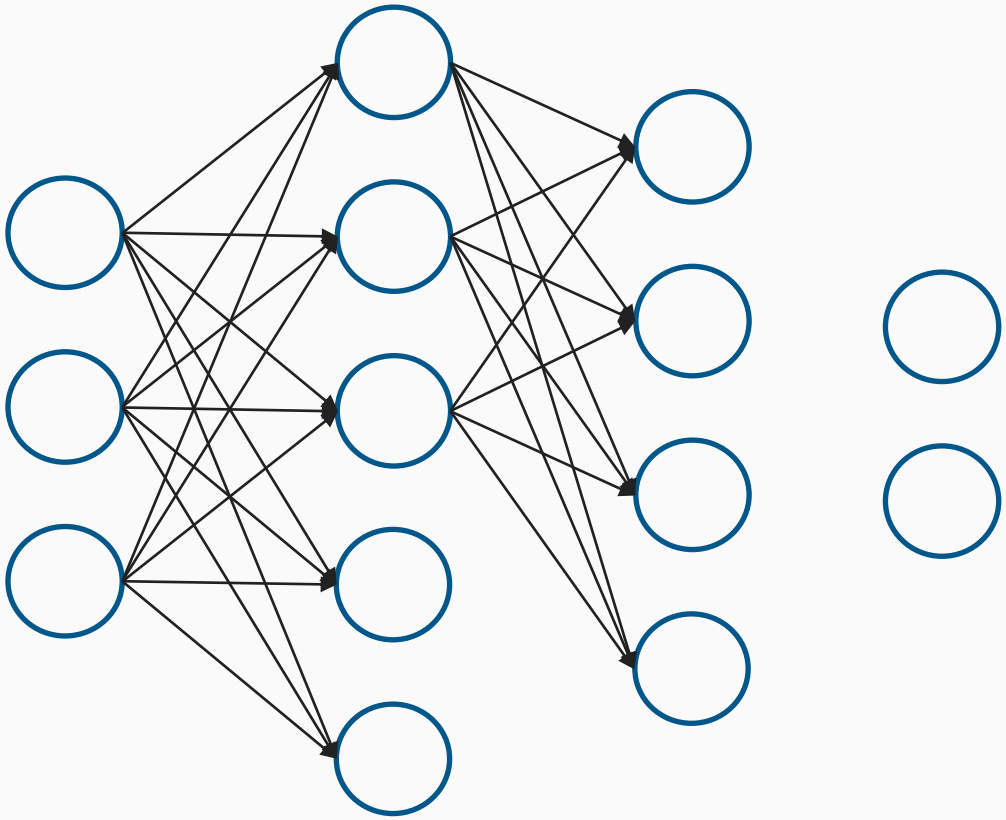


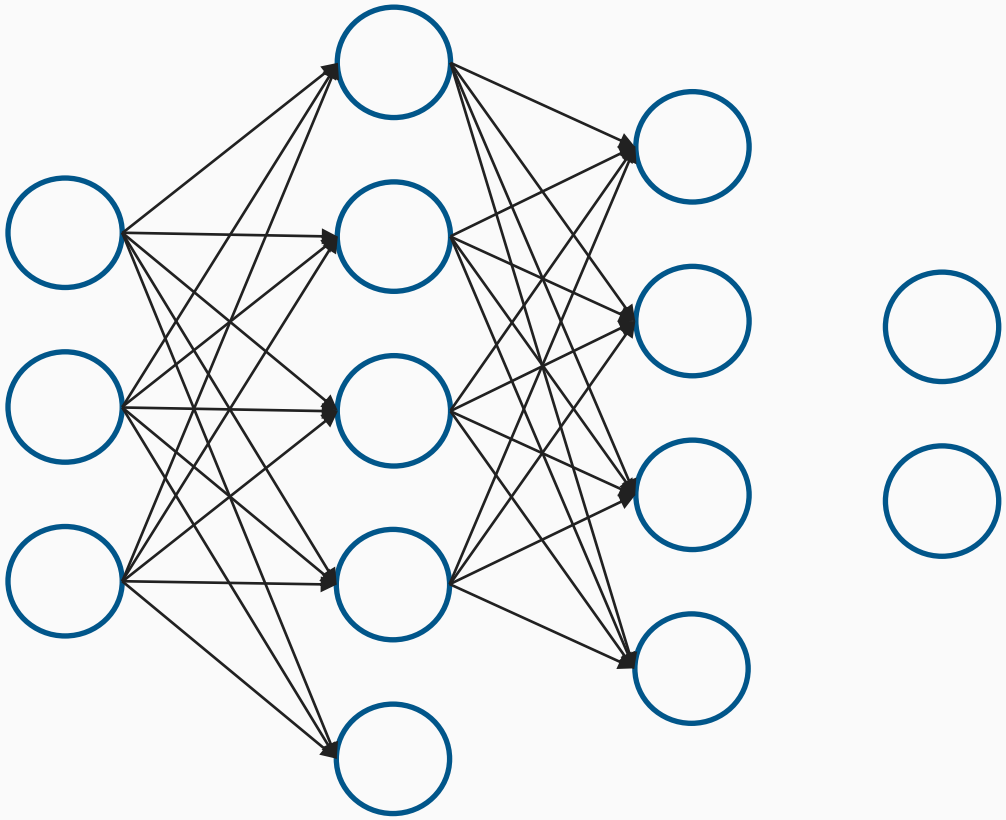


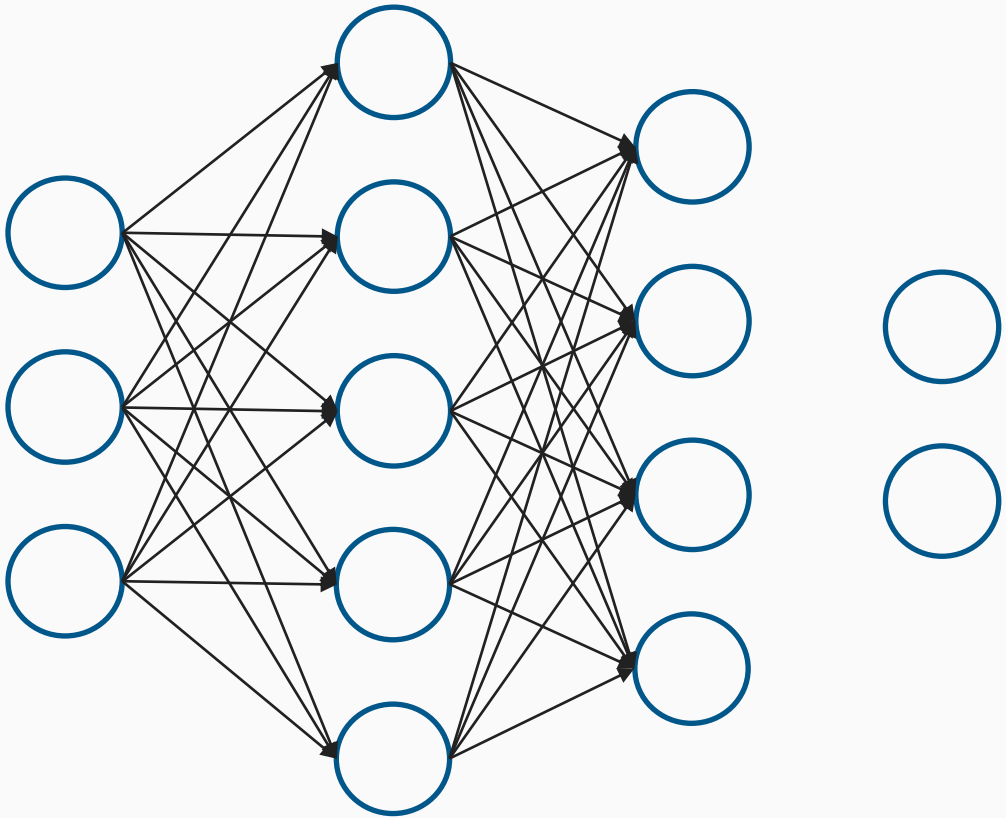


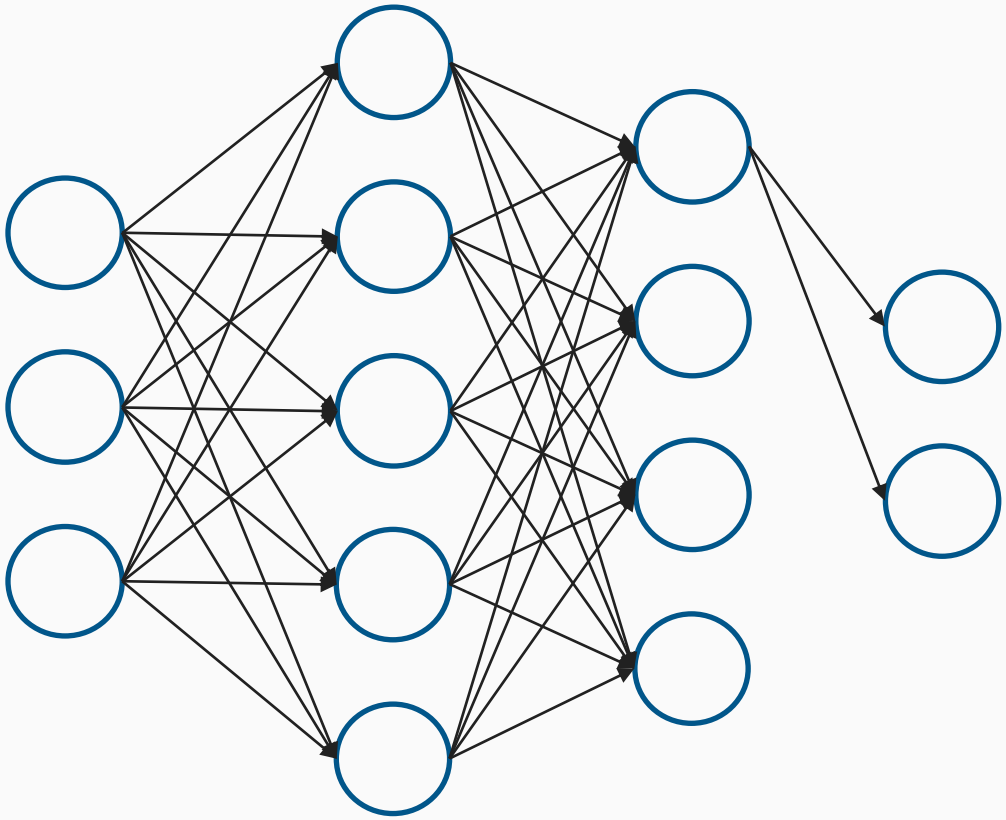


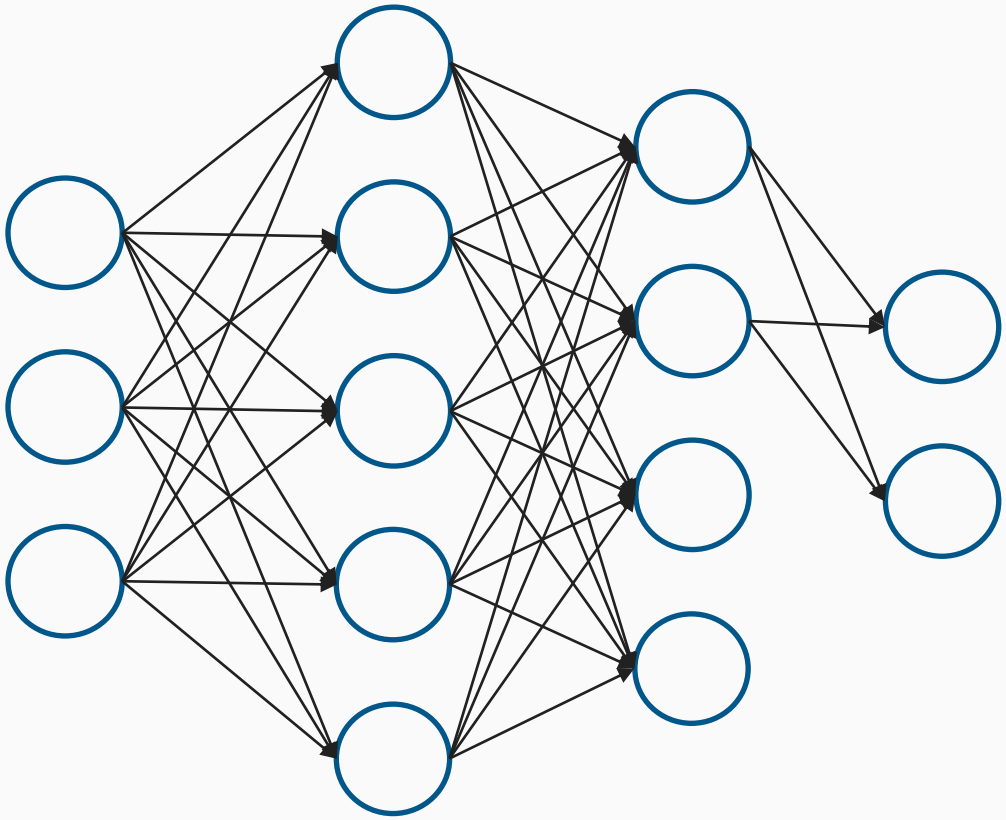


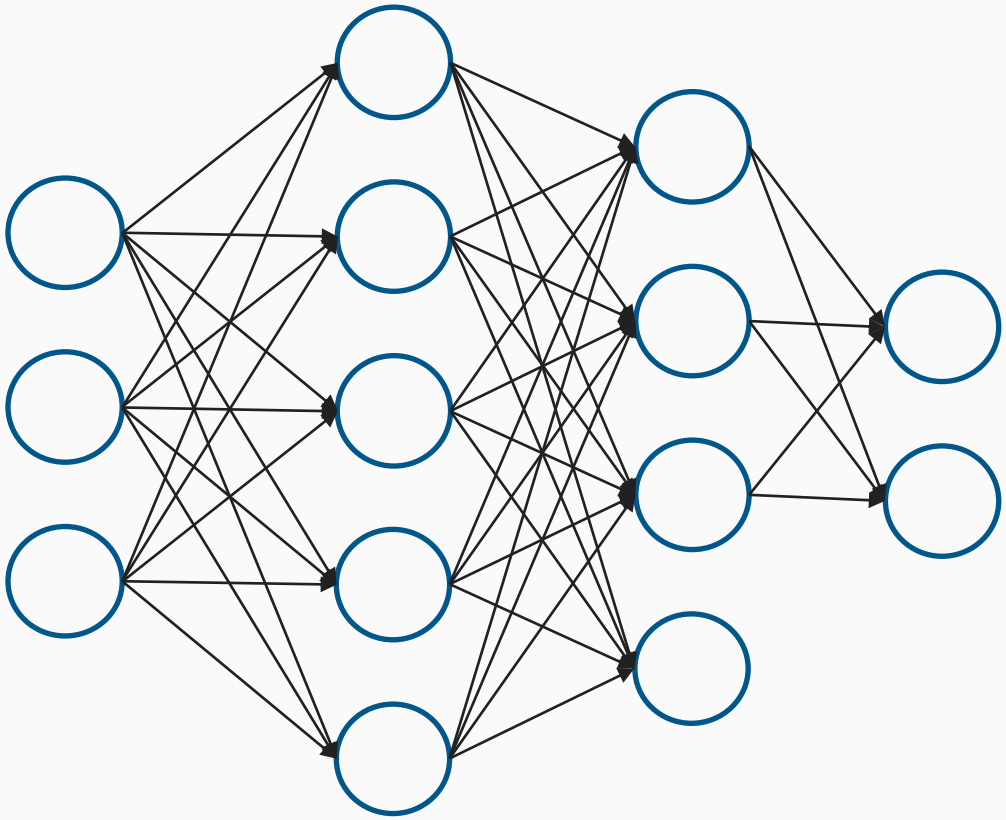


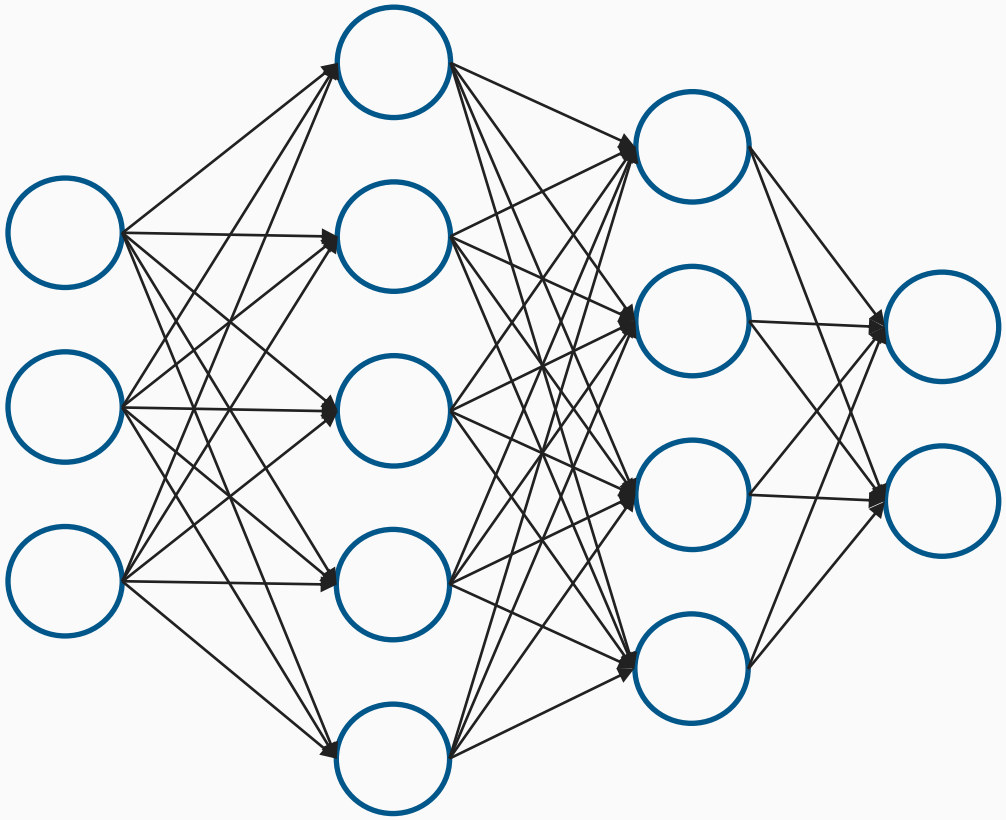


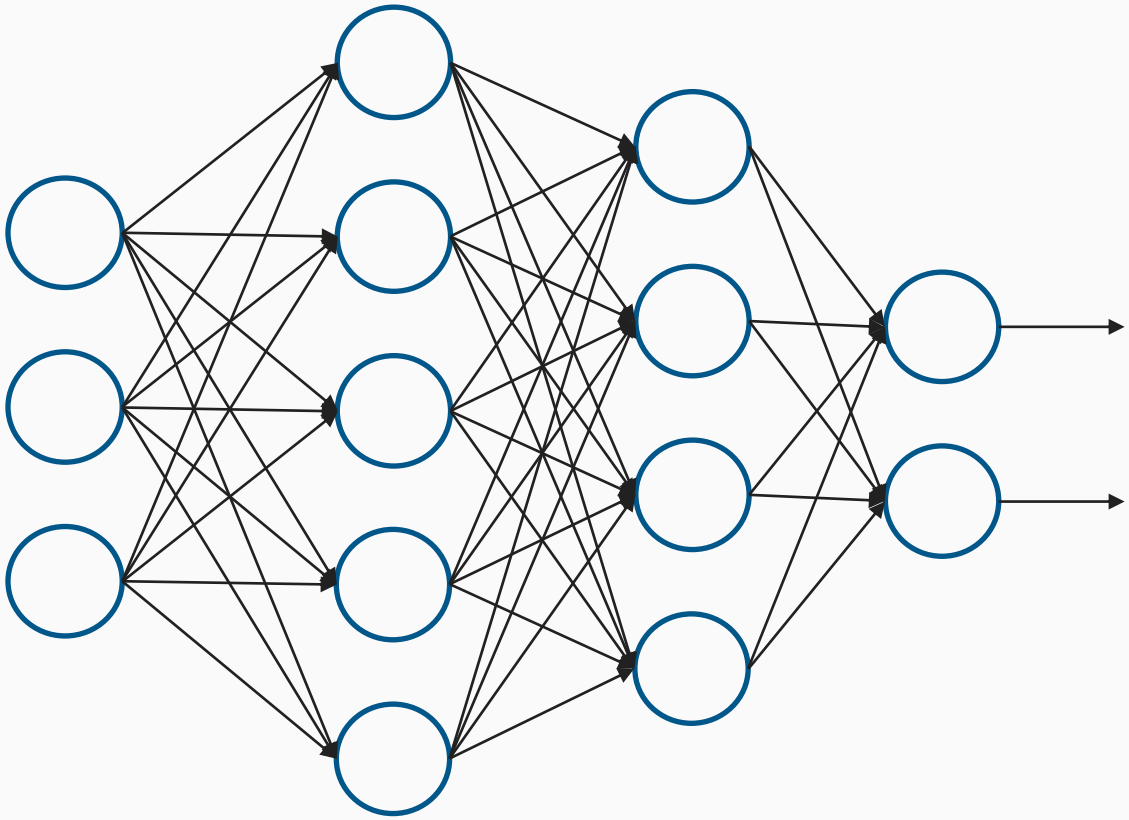


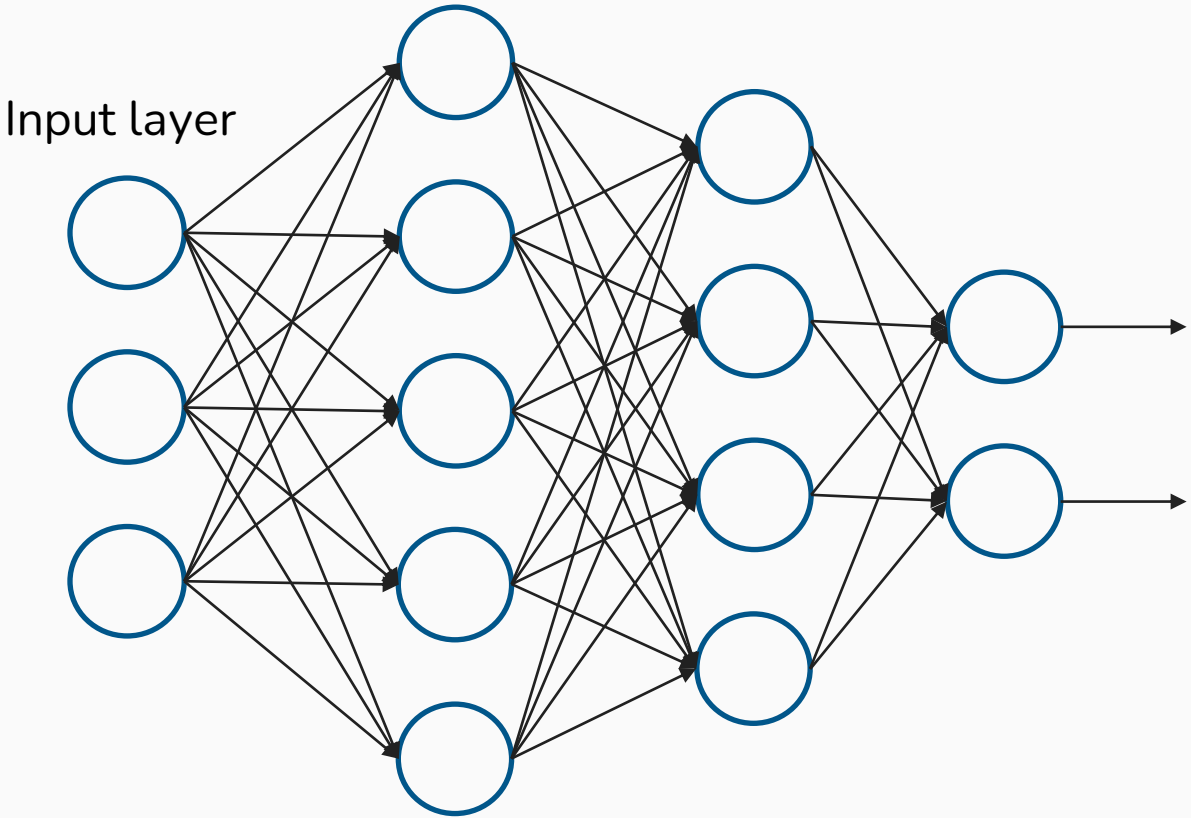


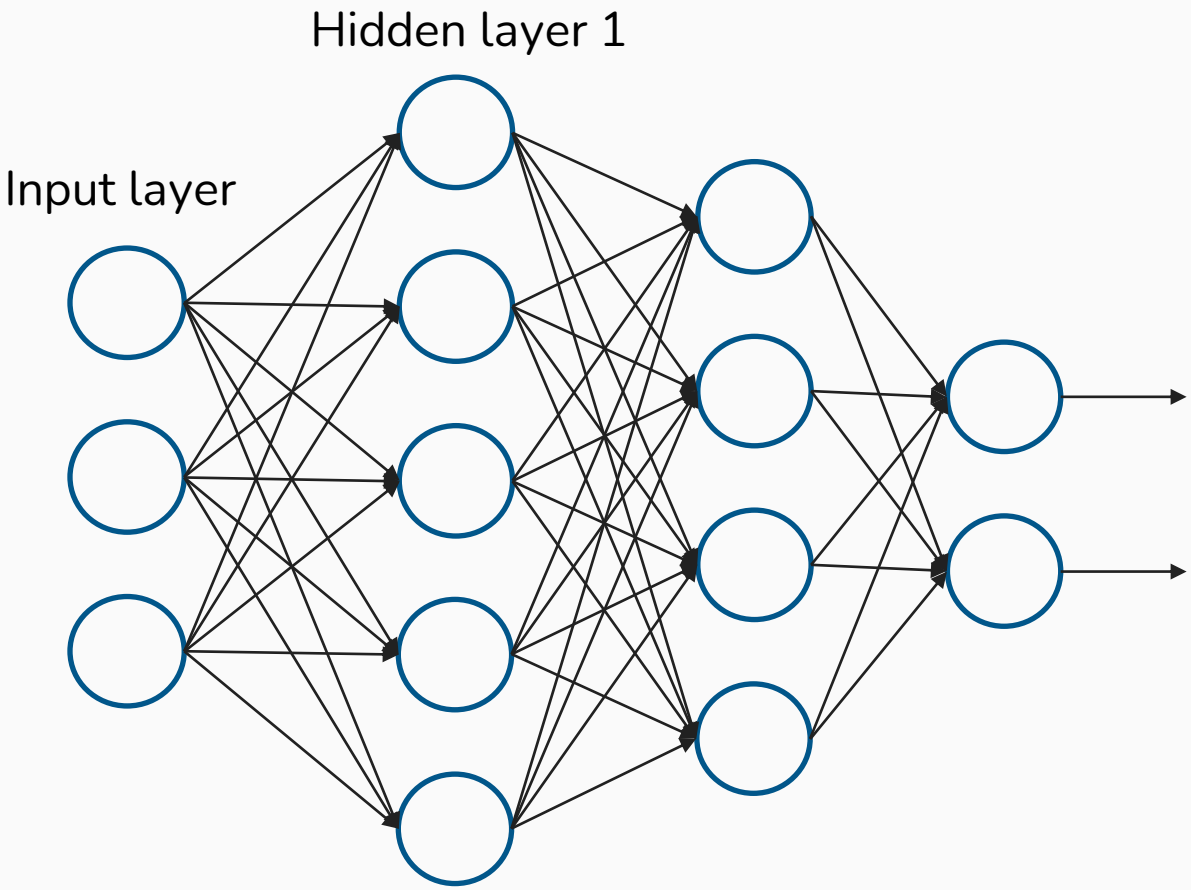


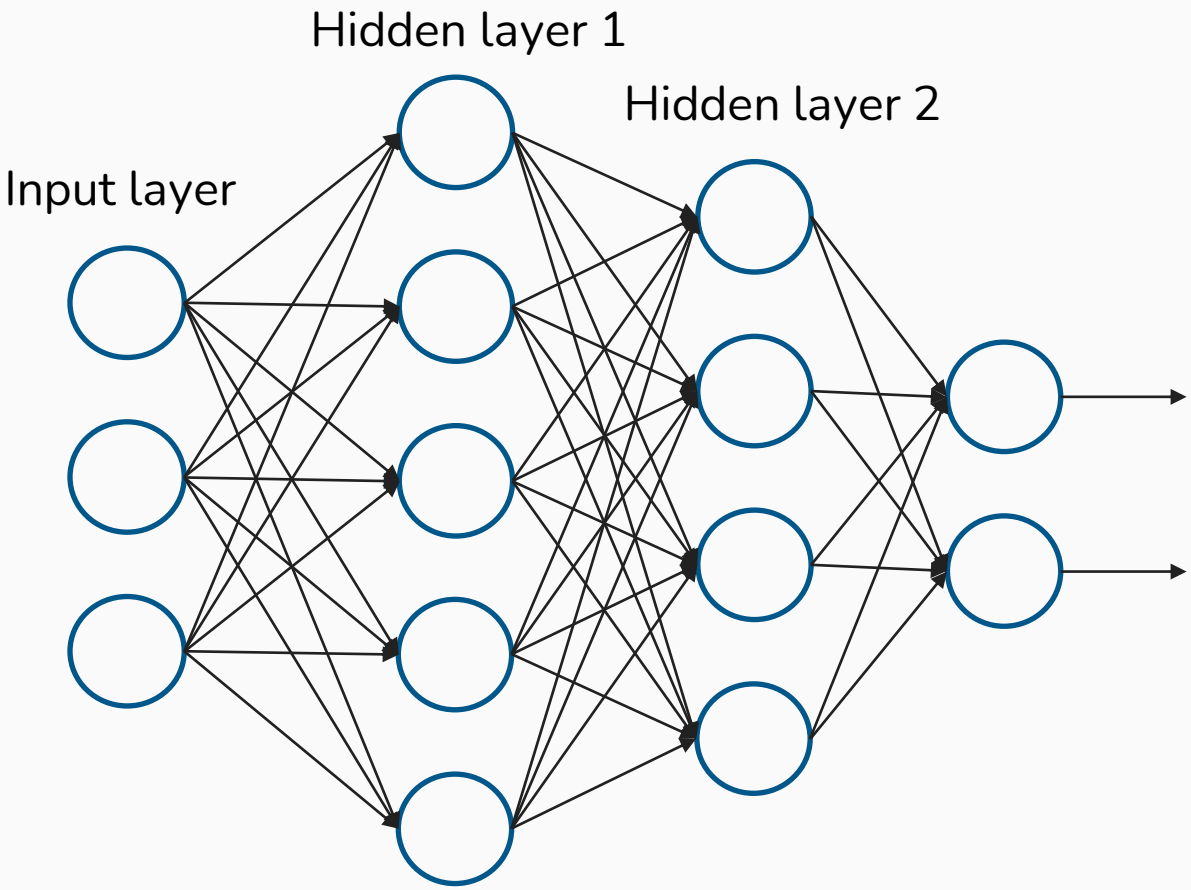


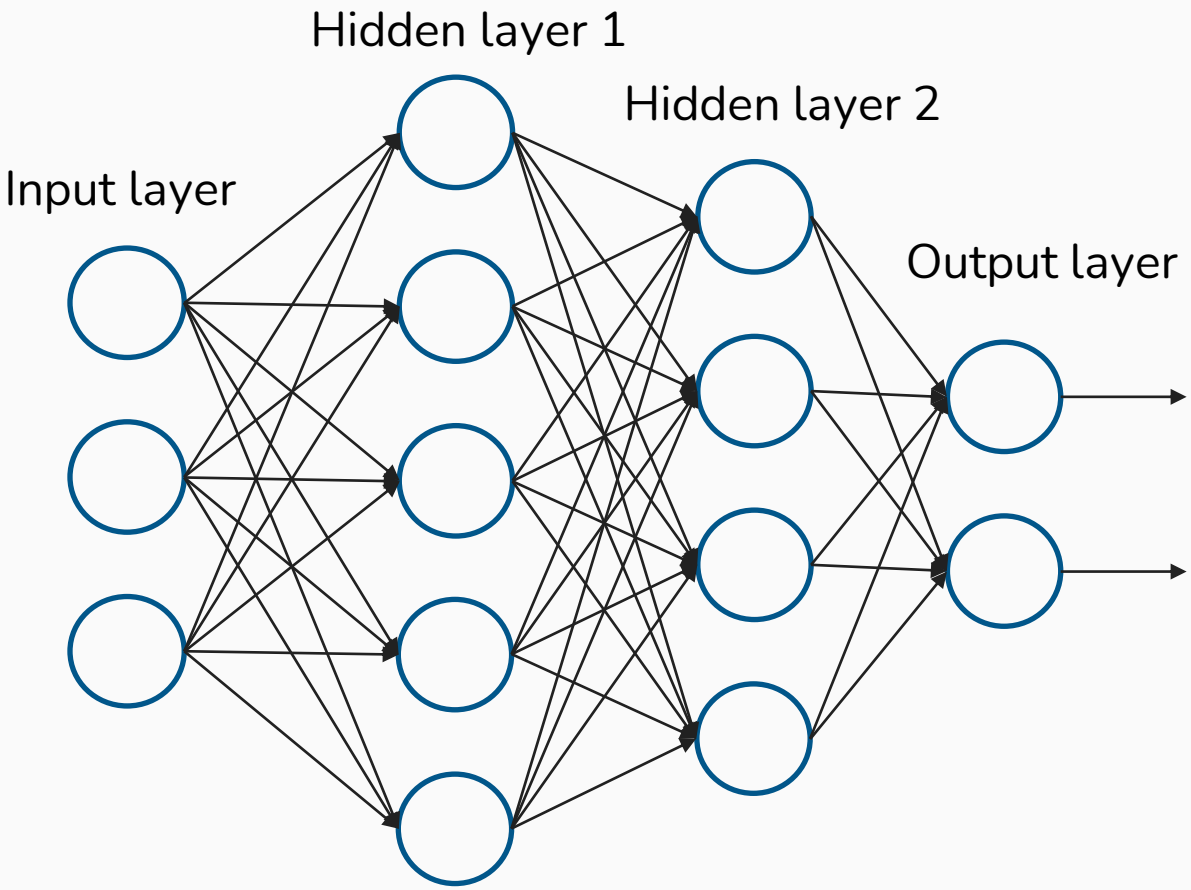


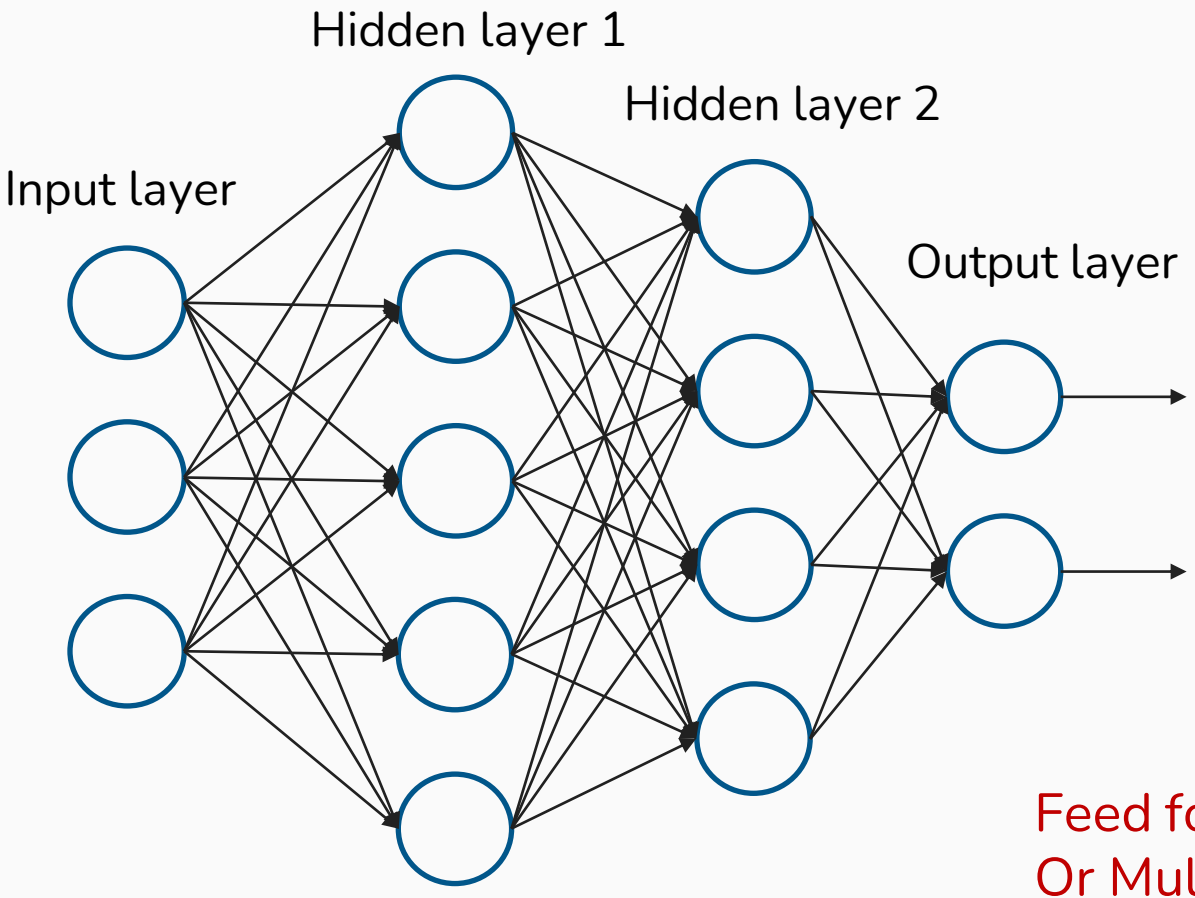



Input layer


Hidden layer 1
Input layer


Hidden layer 1
Hidden layer 2
Input layer


Hidden layer 1
Hidden layer 2
Input layer
Output layer



Feed forward NNs
Or Multi-Layer Perceptron

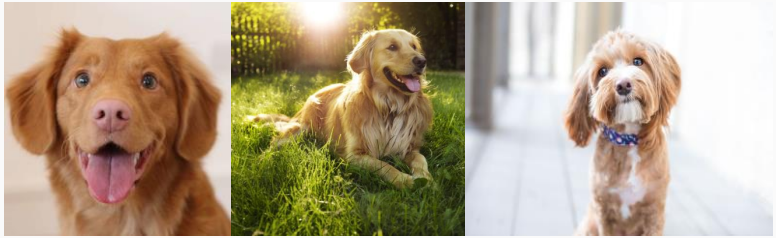
The goal of FFNN:

The goal of FFNN:

Approximate some function f^*

Example :

dogs

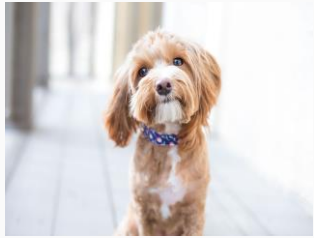


cats

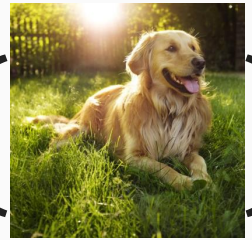
$$= f^*(\quad)$$

Example :

dogs

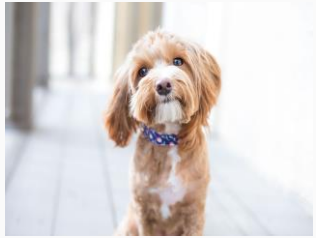


cats

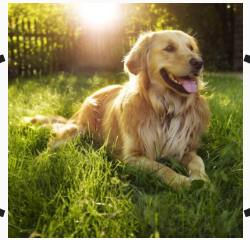
$= f^*($  $)$

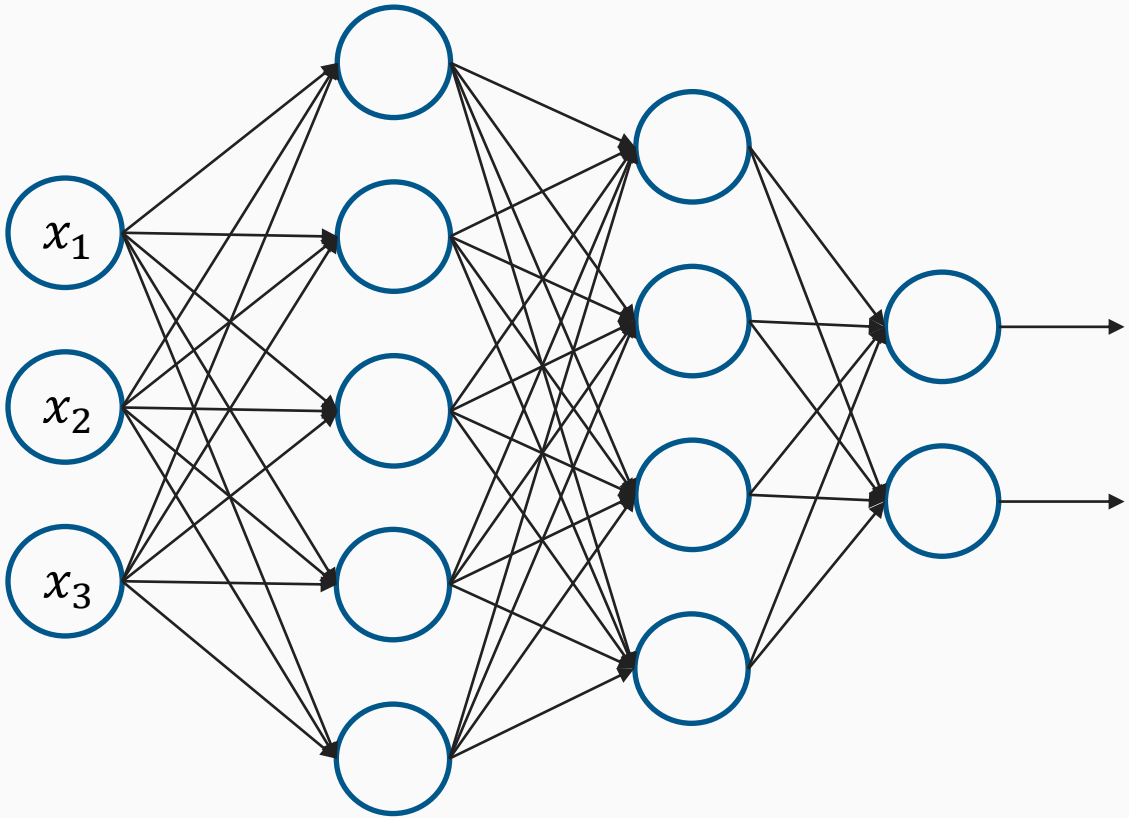
Example :

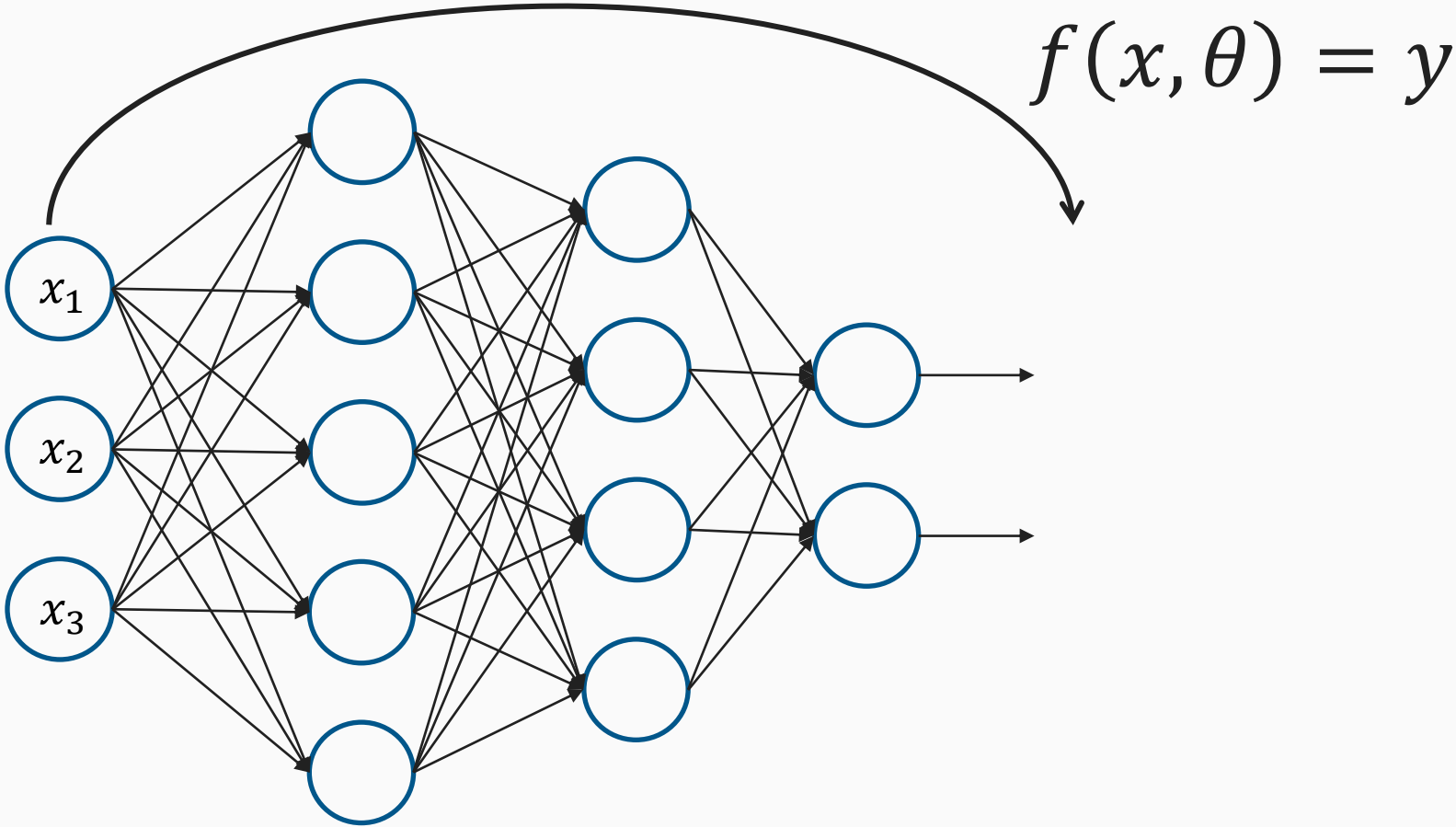
dogs



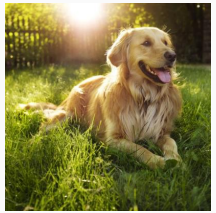
cats

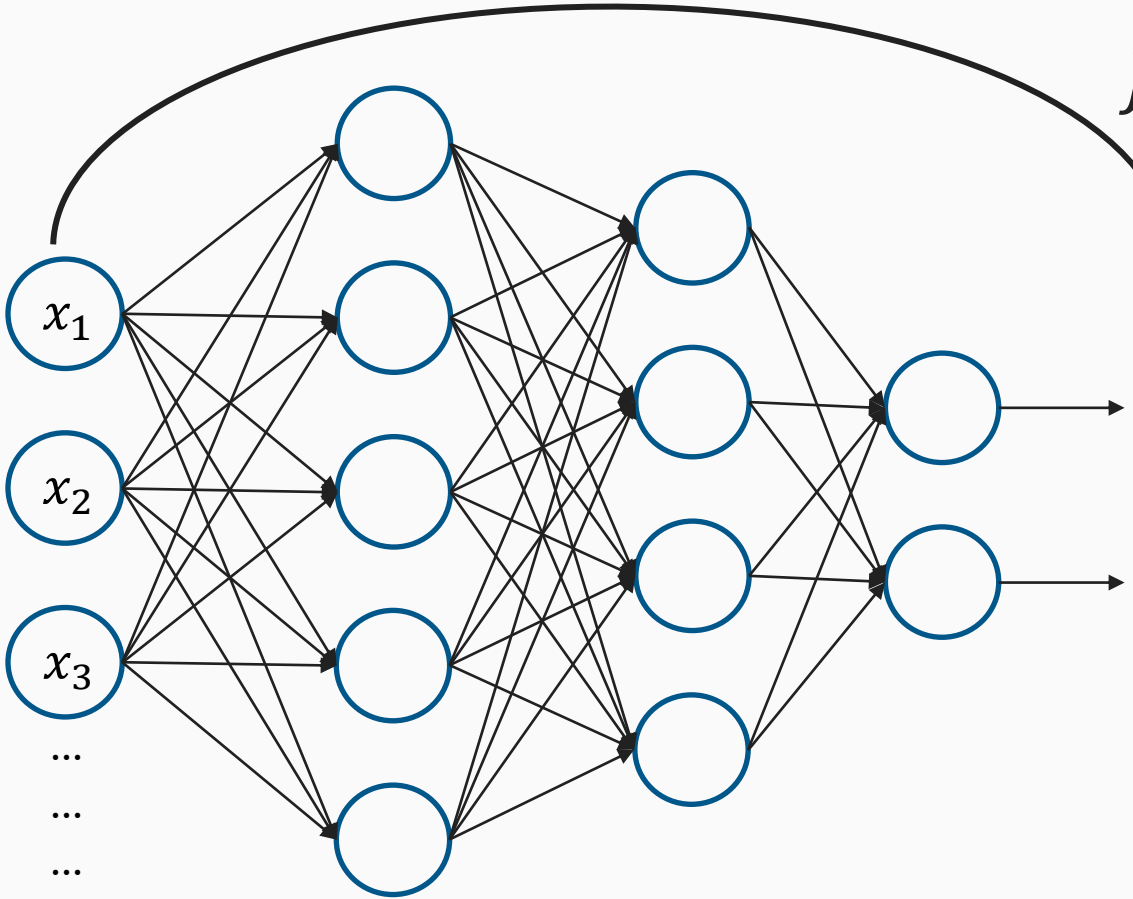
$$dog = f^*($$  $$)$$


x_1
x_2
x_3

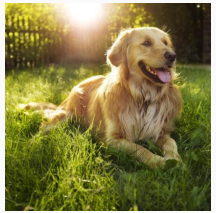

$f(x, \theta) = y$
x_1
x_2
x_3

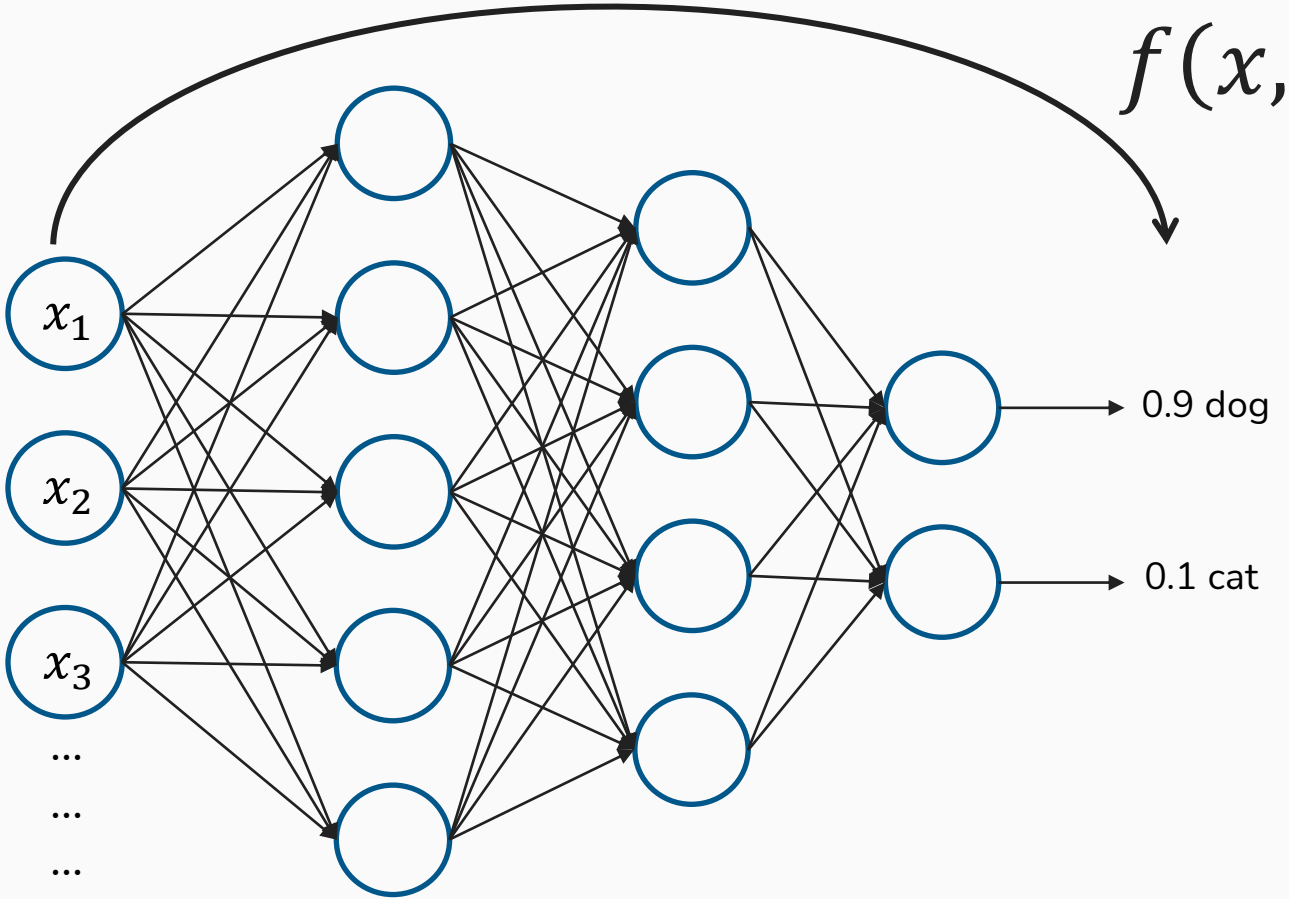
$$f(x, \theta) = y$$

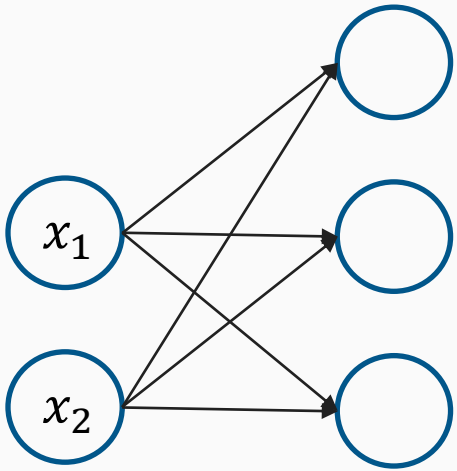


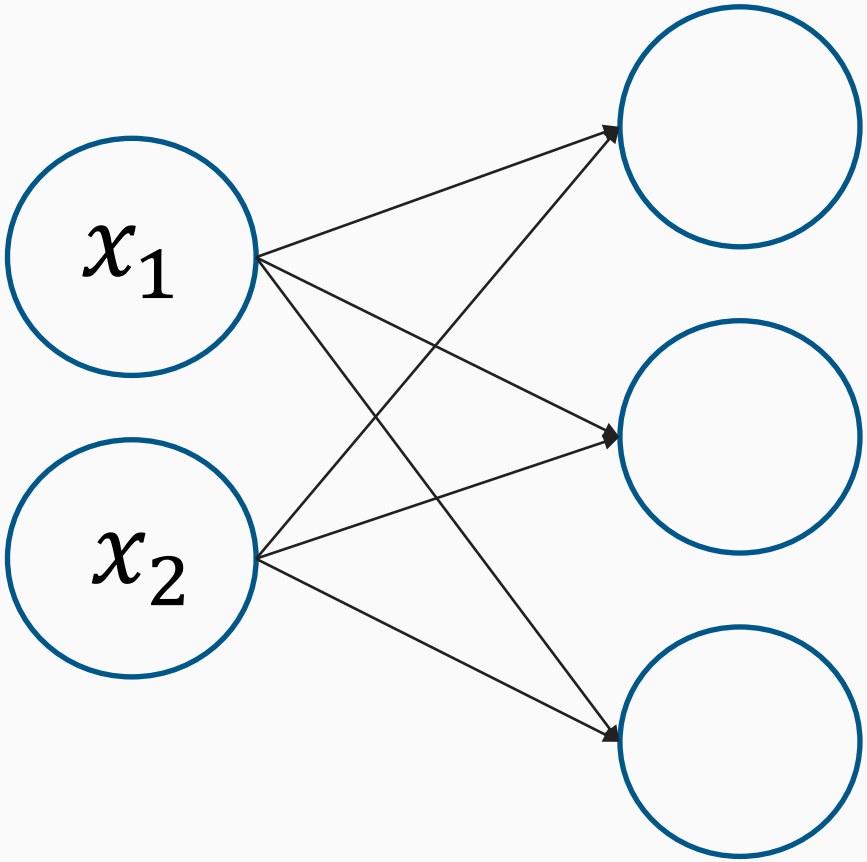


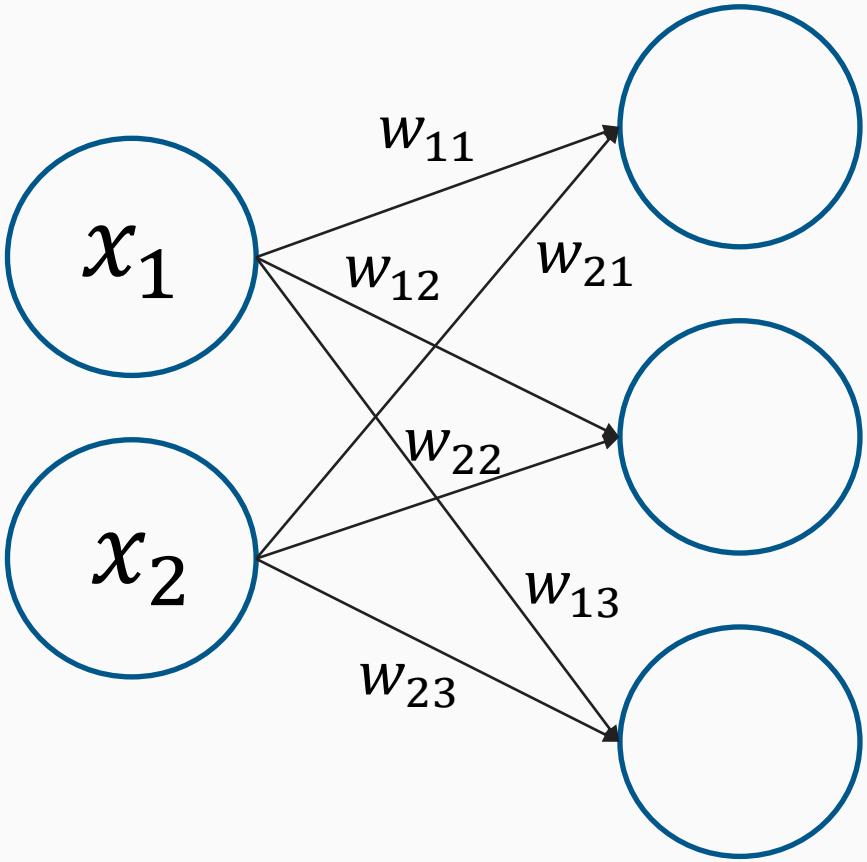
$$f(x, \theta) = y$$

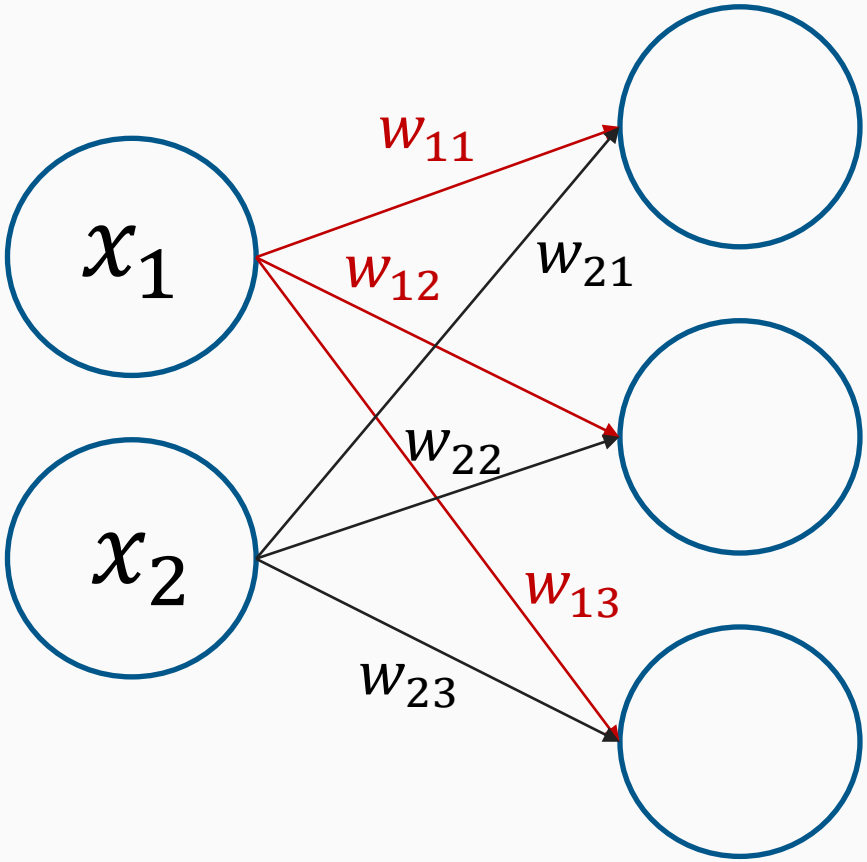


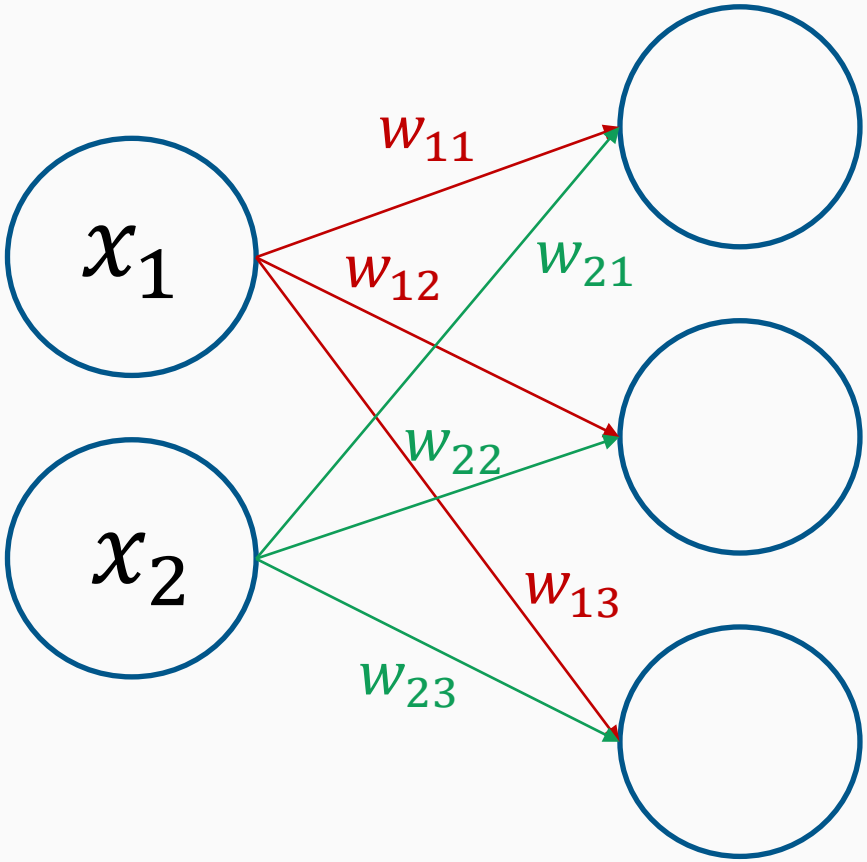


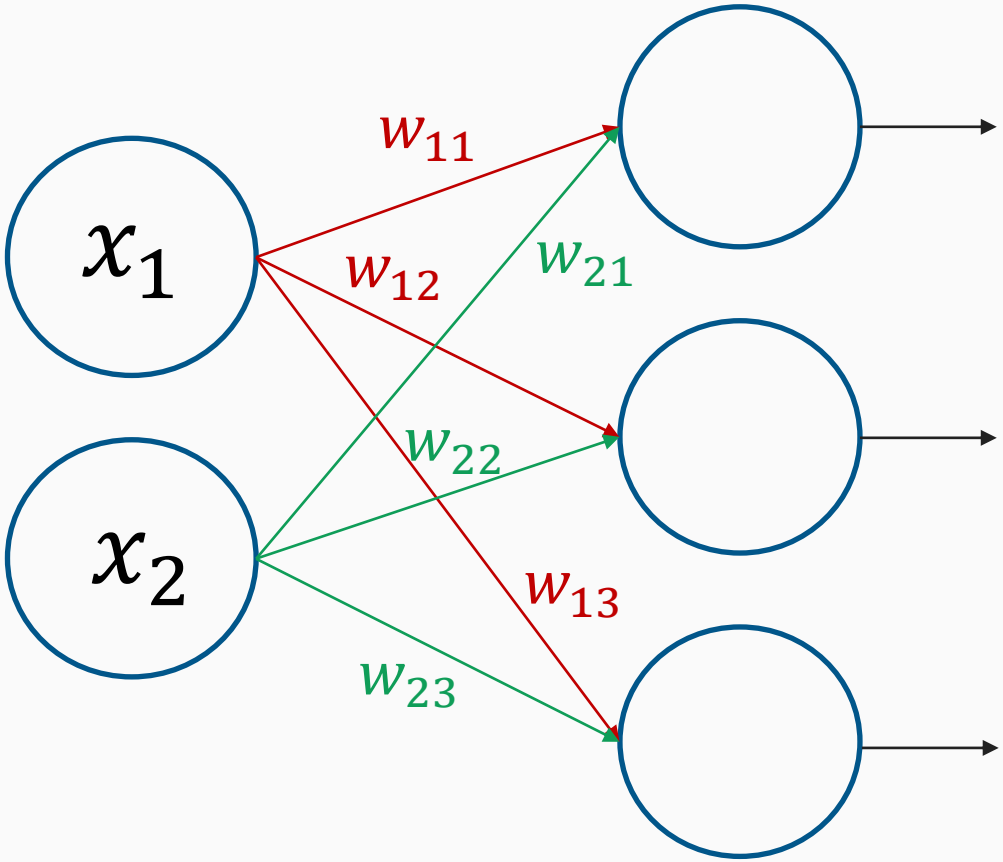

x_1
x_2

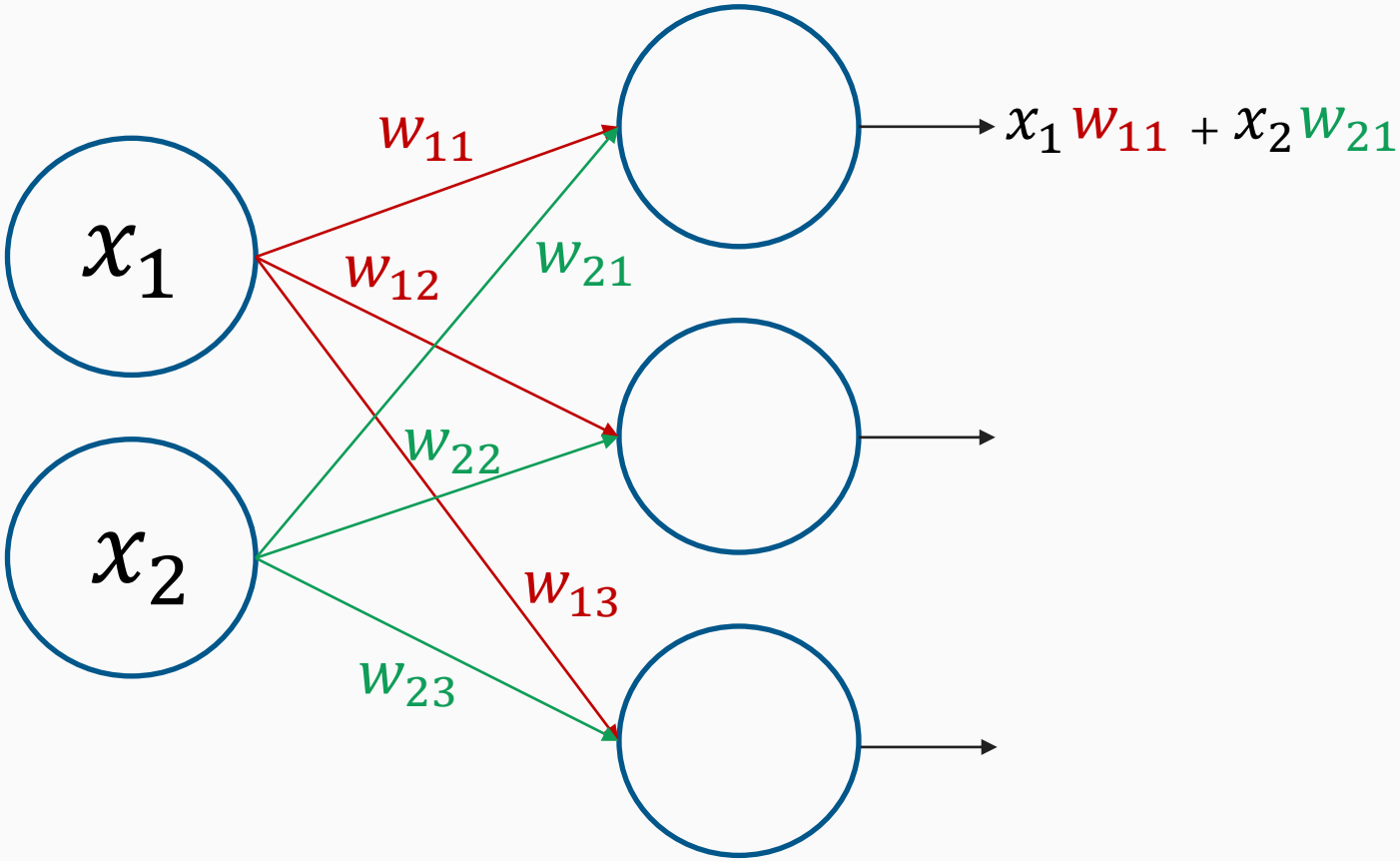

x_1
x_2

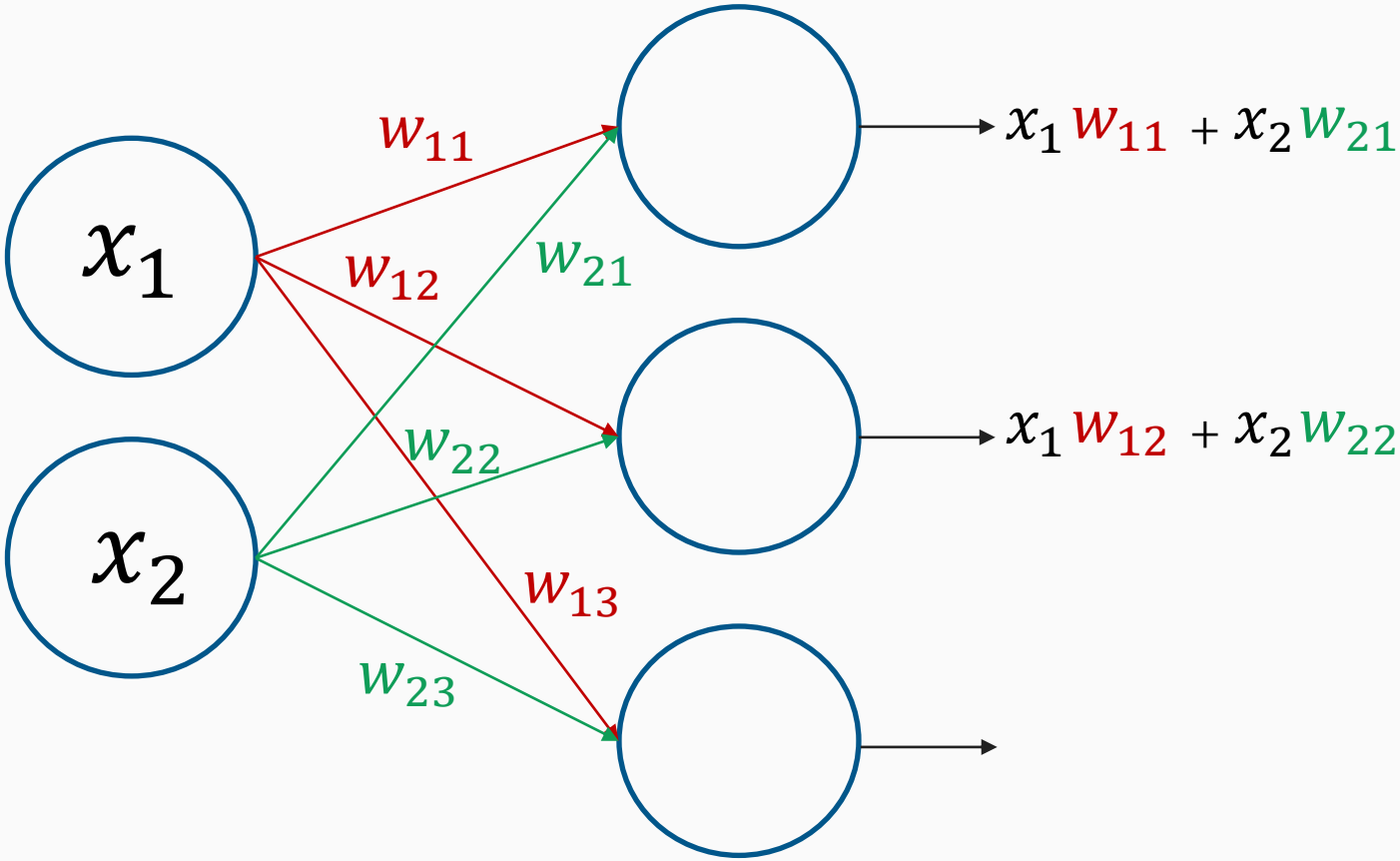

x_1
x_2
w_{11}
w_{12}
w_{21}
w_{22}
w_{13}
w_{23}

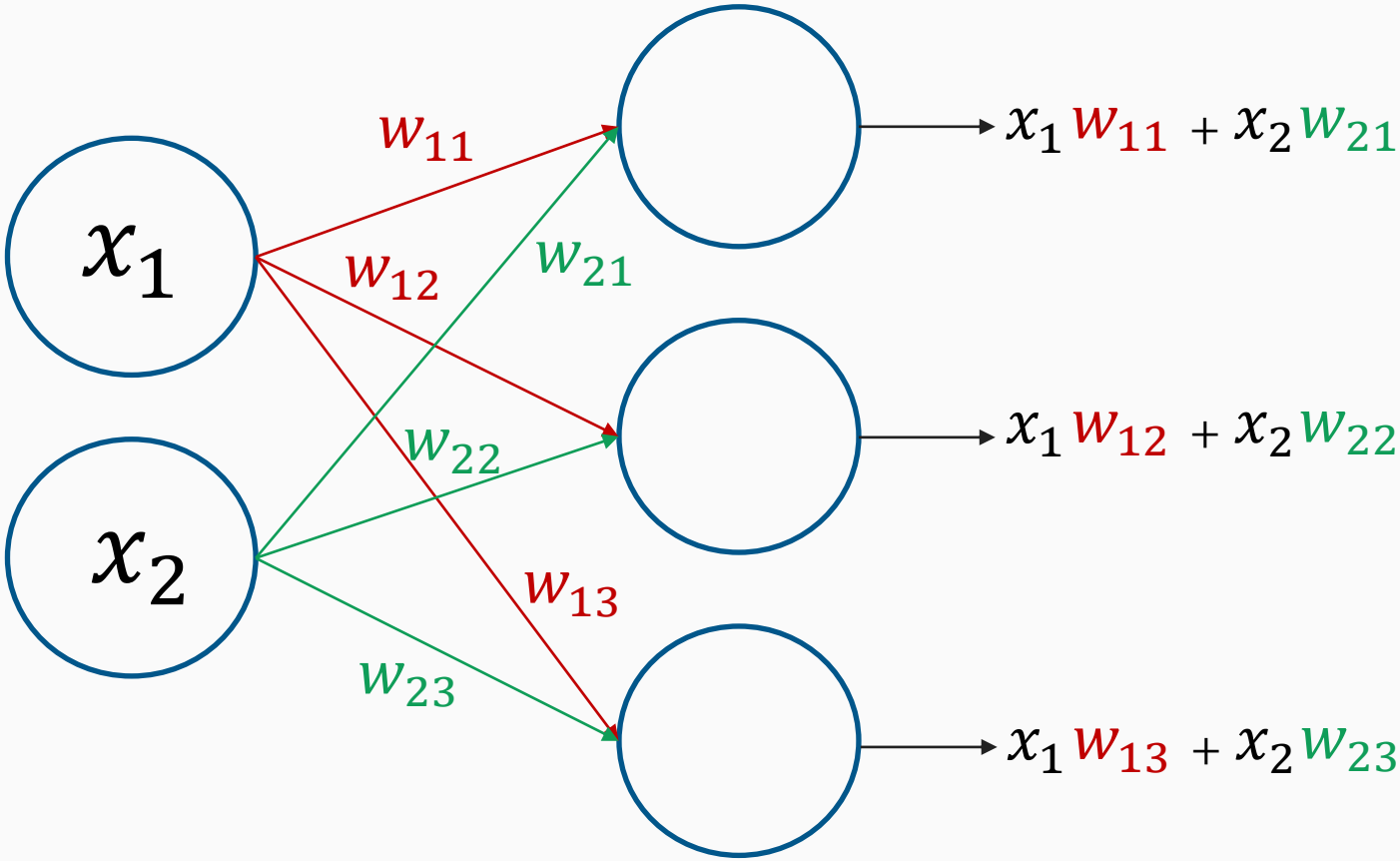

x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}

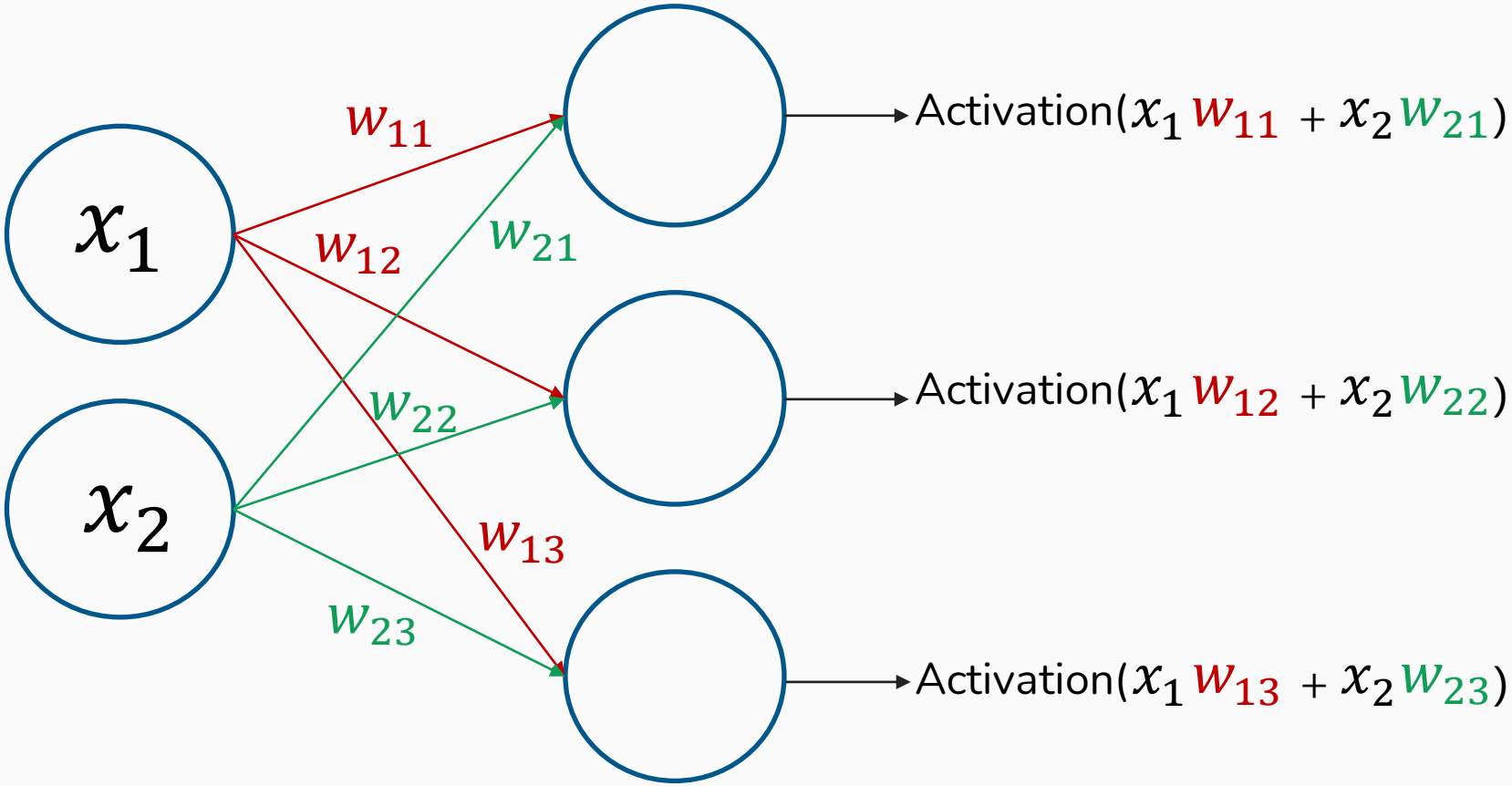

x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}


x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}


x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
$x_1 w_{11} + x_2 w_{21}$

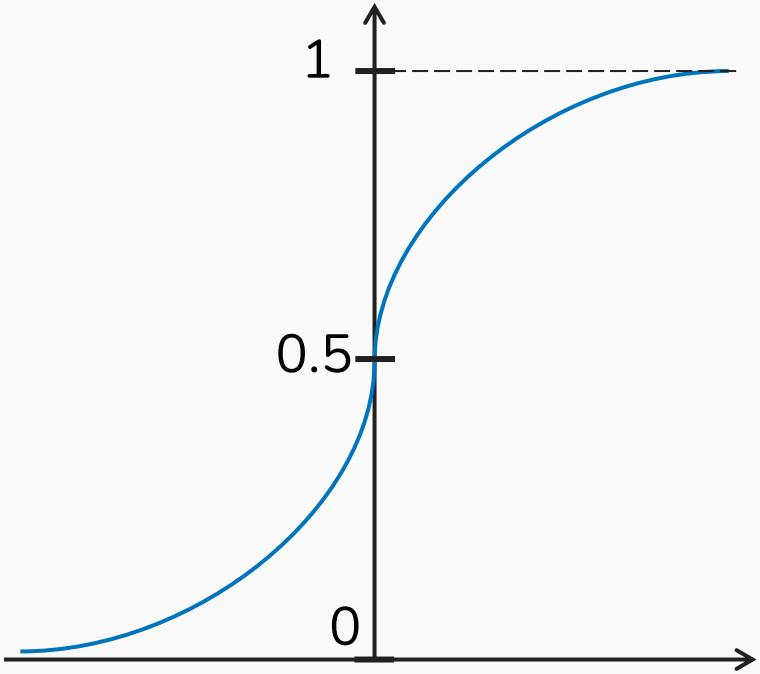

x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
$x_1 w_{11} + x_2 w_{21}$
$x_1 w_{12} + x_2 w_{22}$


x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
$x_1 w_{11} + x_2 w_{21}$
$x_1 w_{12} + x_2 w_{22}$
$x_1 w_{13} + x_2 w_{23}$

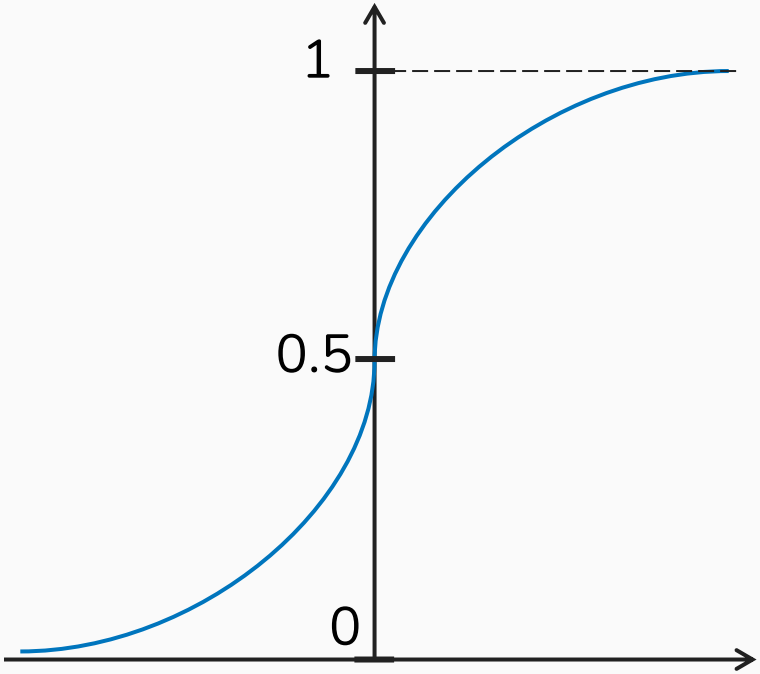

x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
Activation($x_1 w_{11} + x_2 w_{21}$)
Activation($x_1 w_{12} + x_2 w_{22}$)
Activation($x_1 w_{13} + x_2 w_{23}$)

Activation functions

Sigmoid

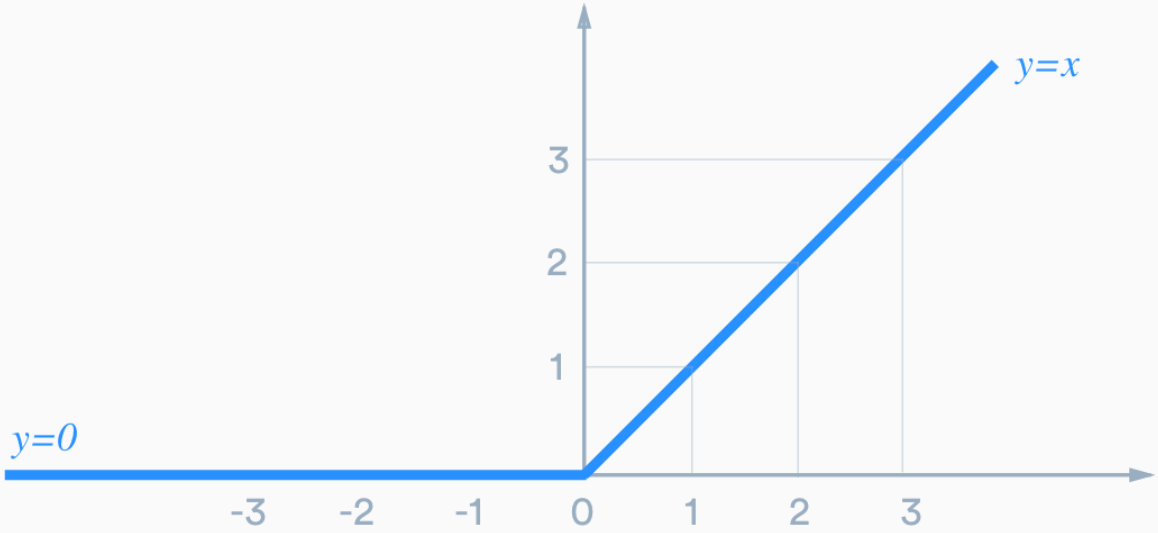


Sigmoid

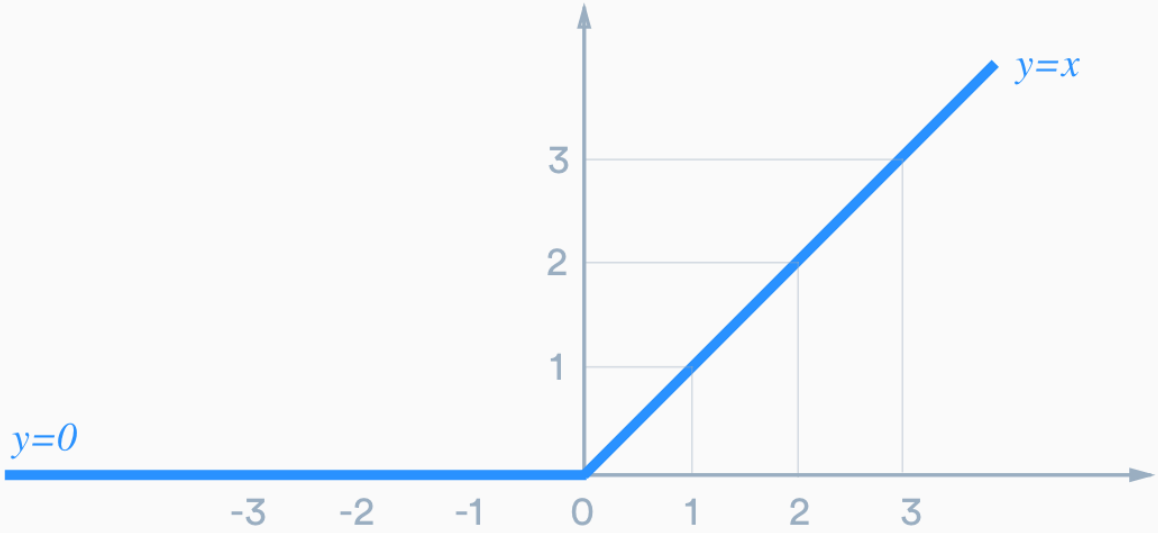


$$s(z) = \frac{1}{1 + e^{-z}}$$

ReLU



ReLU



$$ReLU(x) = max(0, x)$$

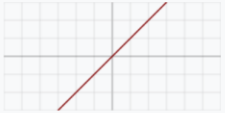
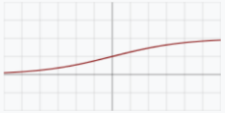
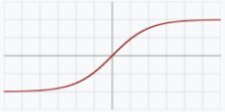
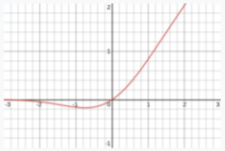
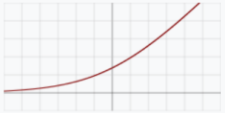
Softmax

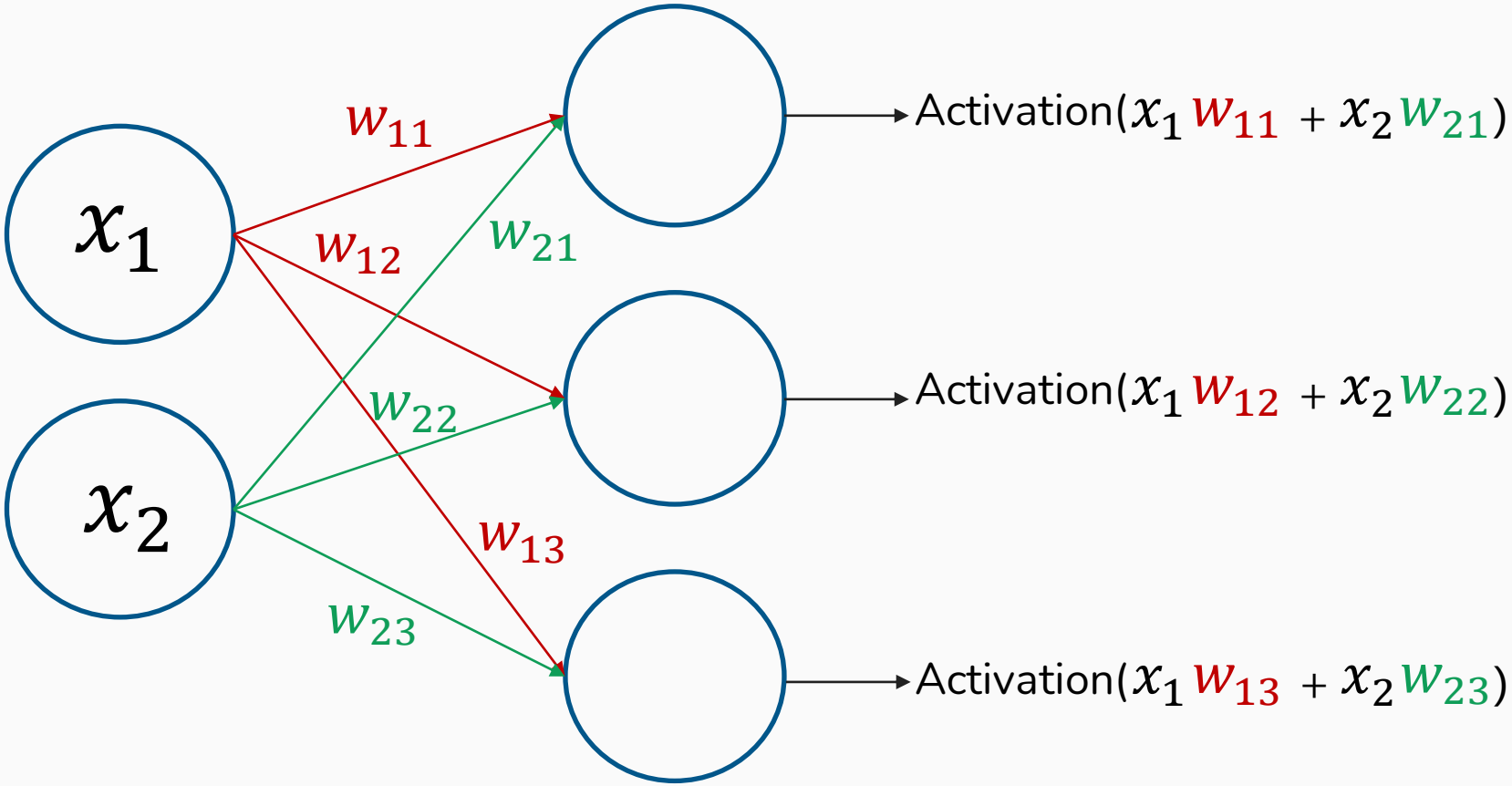
$$\sigma_i(h) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

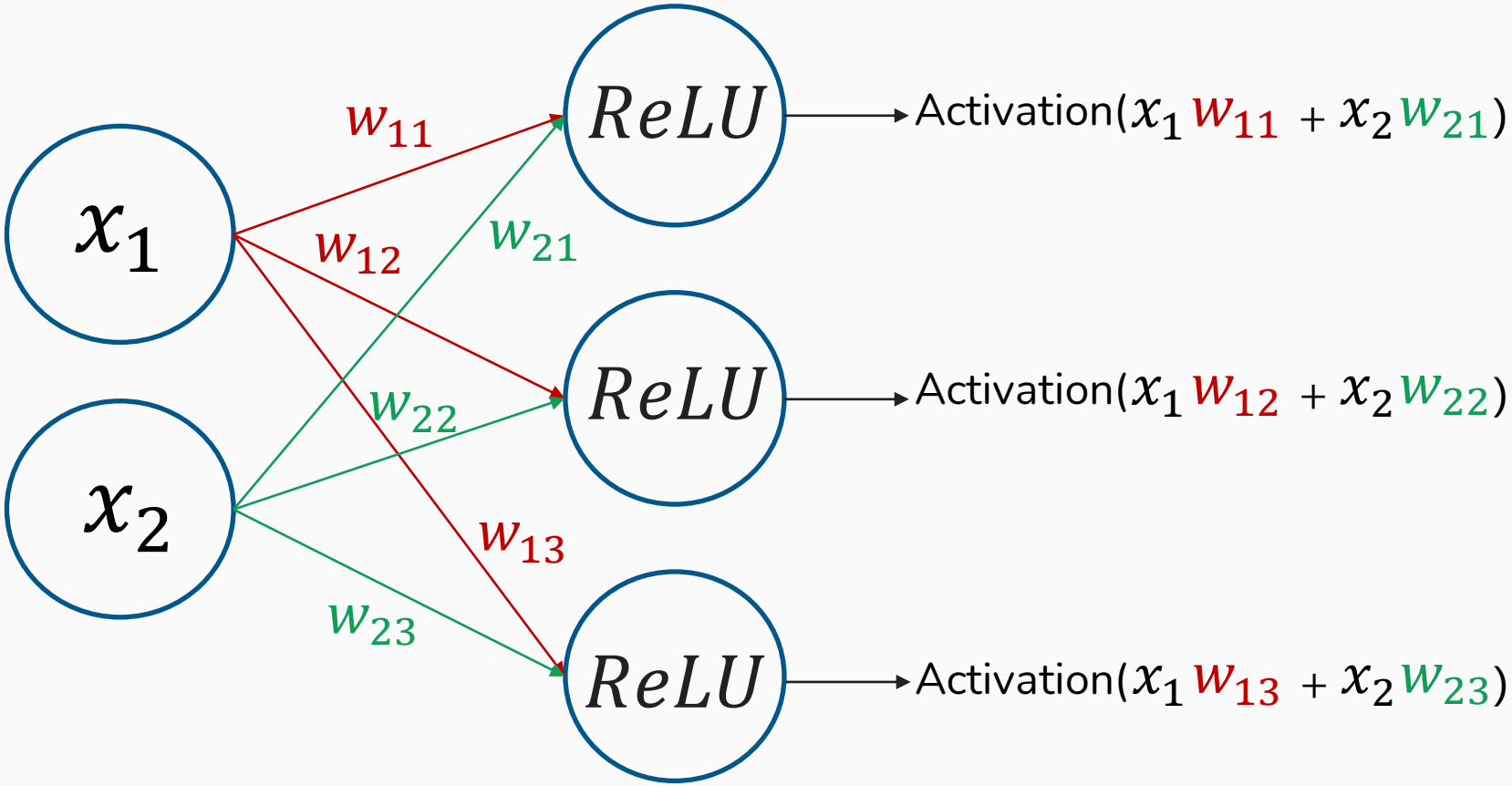
Softmax

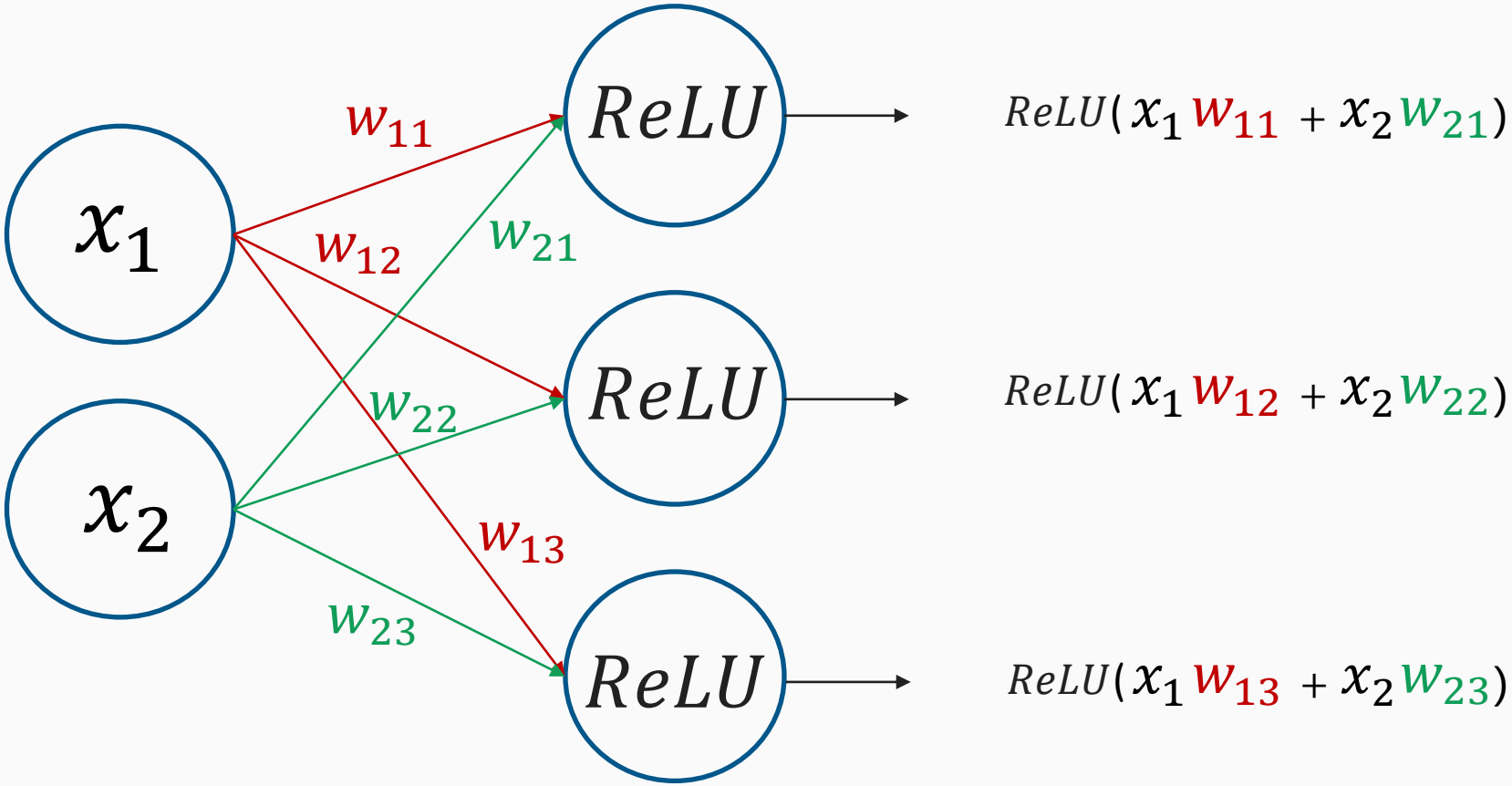
$$\sigma_i(h) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

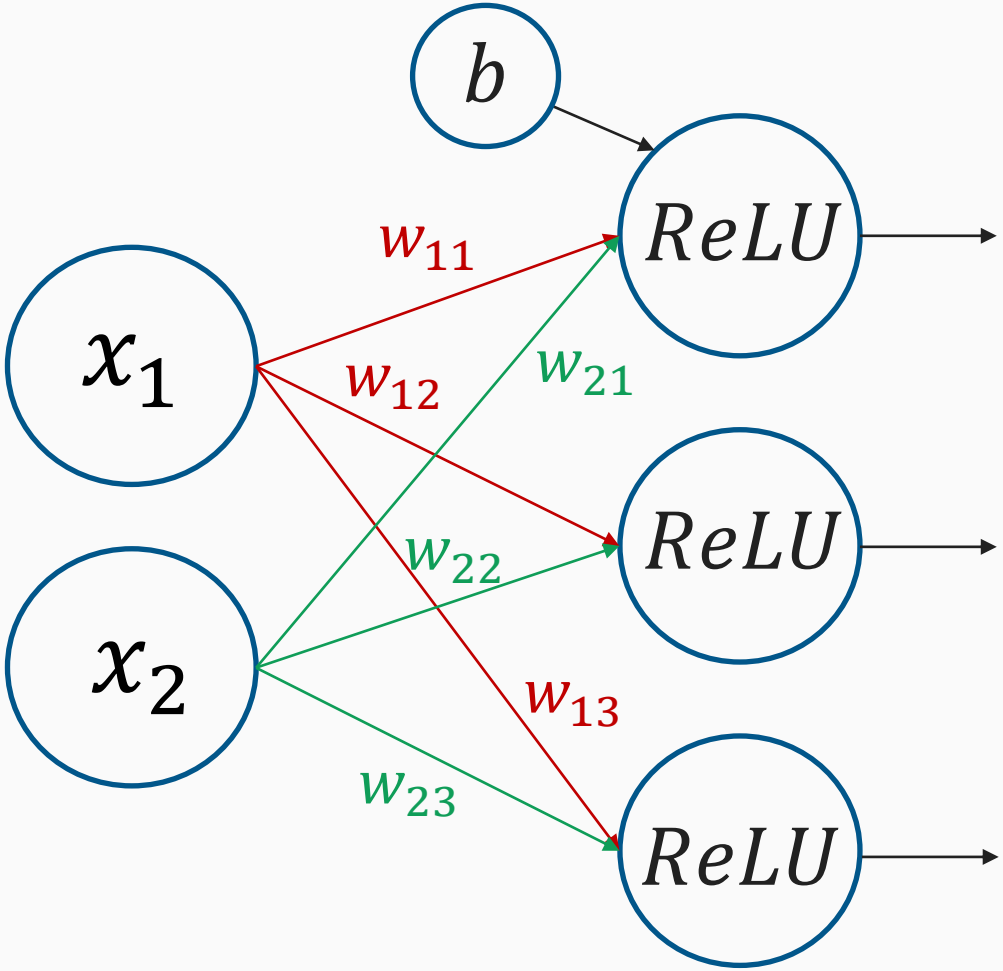
Transform outputs into probablities

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range	Orde contin
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$[1]	$f(x)(1-f(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[8]		$\ln(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
Exponential		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$	$\begin{cases} \alpha e^x & \text{if } x < 0 \end{cases}$		


x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
Activation($x_1 w_{11} + x_2 w_{21}$)
Activation($x_1 w_{12} + x_2 w_{22}$)
Activation($x_1 w_{13} + x_2 w_{23}$)


x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
$ReLU$
$ReLU$
$ReLU$
Activation($x_1 w_{11} + x_2 w_{21}$)
Activation($x_1 w_{12} + x_2 w_{22}$)
Activation($x_1 w_{13} + x_2 w_{23}$)

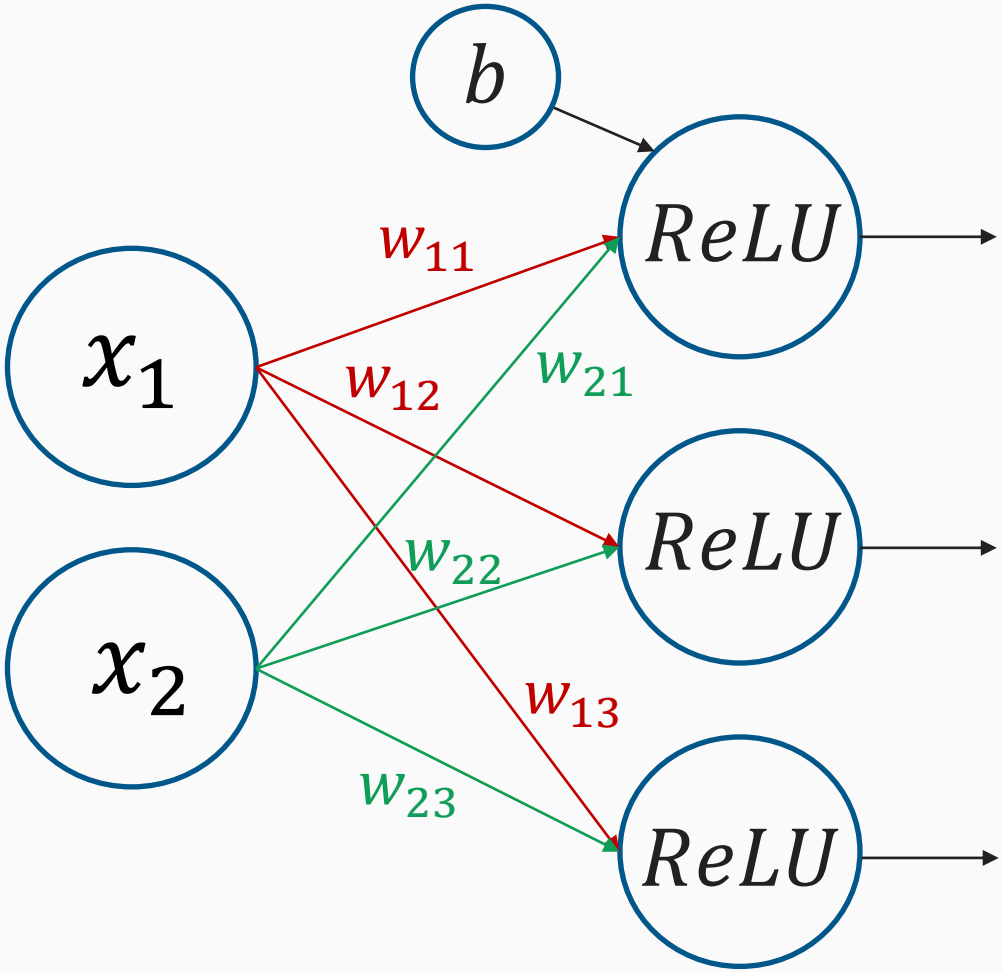

x_1
x_2
w_{11}
w_{12}
w_{13}
w_{21}
w_{22}
w_{23}
$ReLU$
$ReLU$
$ReLU$
$ReLU(x_1 w_{11} + x_2 w_{21})$
$ReLU(x_1 w_{12} + x_2 w_{22})$
$ReLU(x_1 w_{13} + x_2 w_{23})$



$ReLU(x_1 w_{11} + x_2 w_{21})$

$ReLU(x_1 w_{12} + x_2 w_{22})$

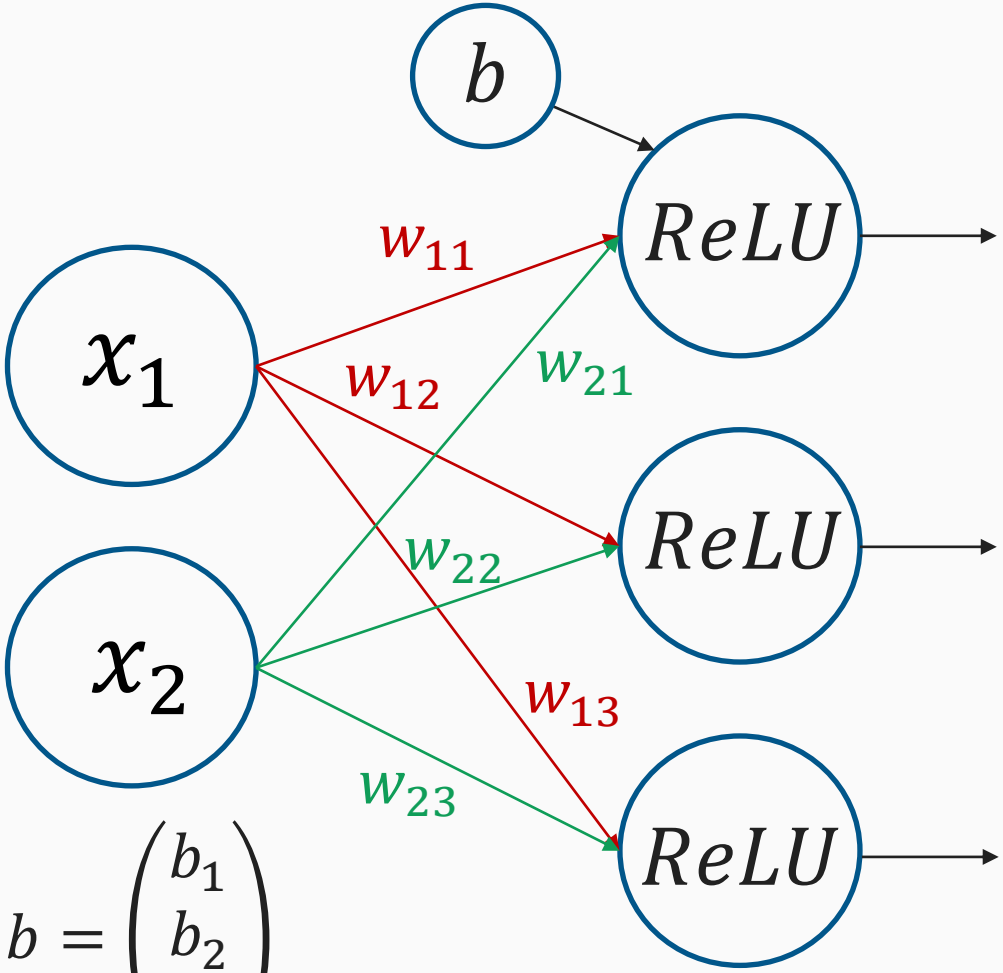
$ReLU(x_1 w_{13} + x_2 w_{23})$



$$ReLU(x_1 w_{11} + x_2 w_{21} + b_1)$$

$$ReLU(x_1 w_{12} + x_2 w_{22} + b_2)$$

$$ReLU(x_1 w_{13} + x_2 w_{23} + b_3)$$

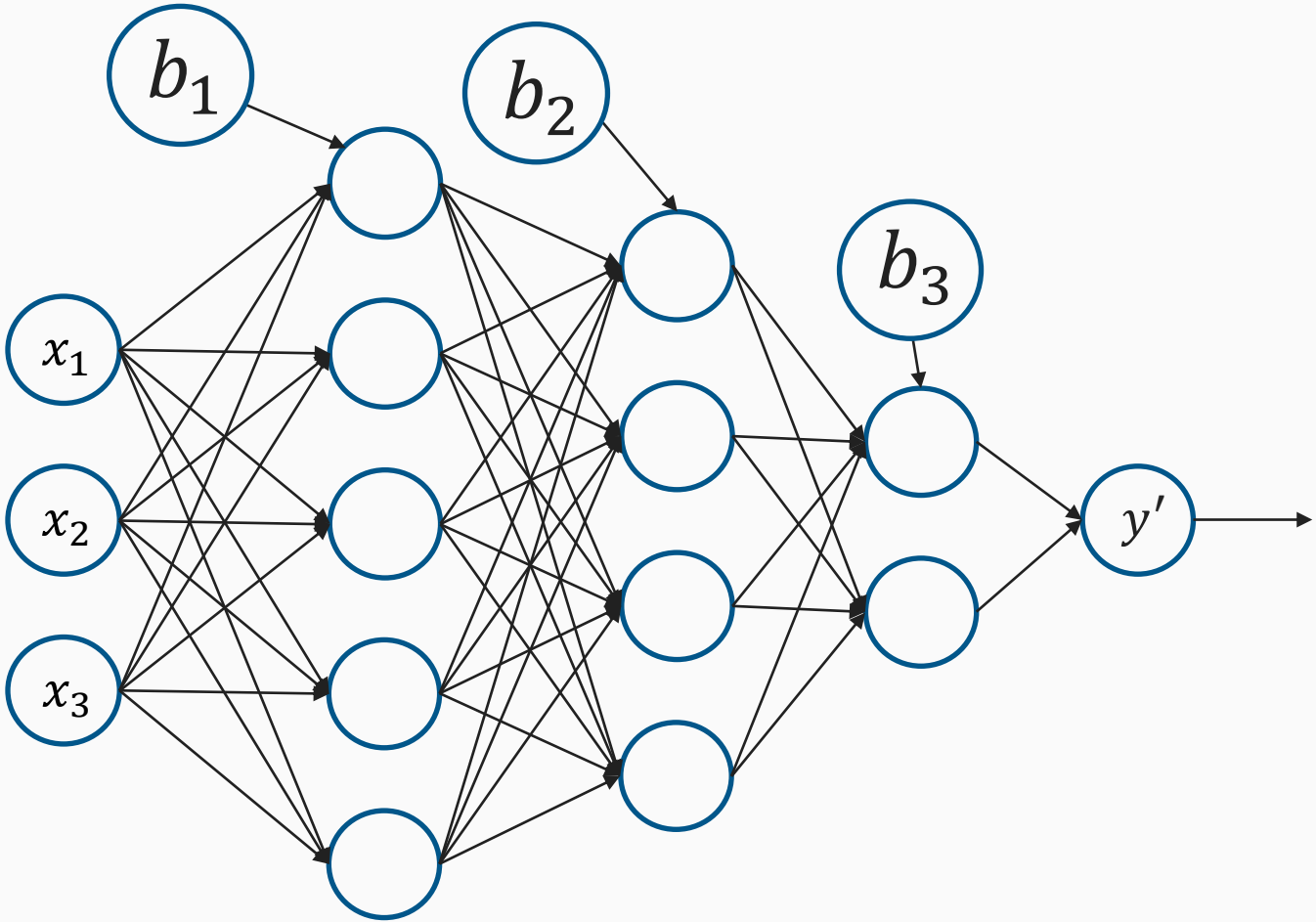


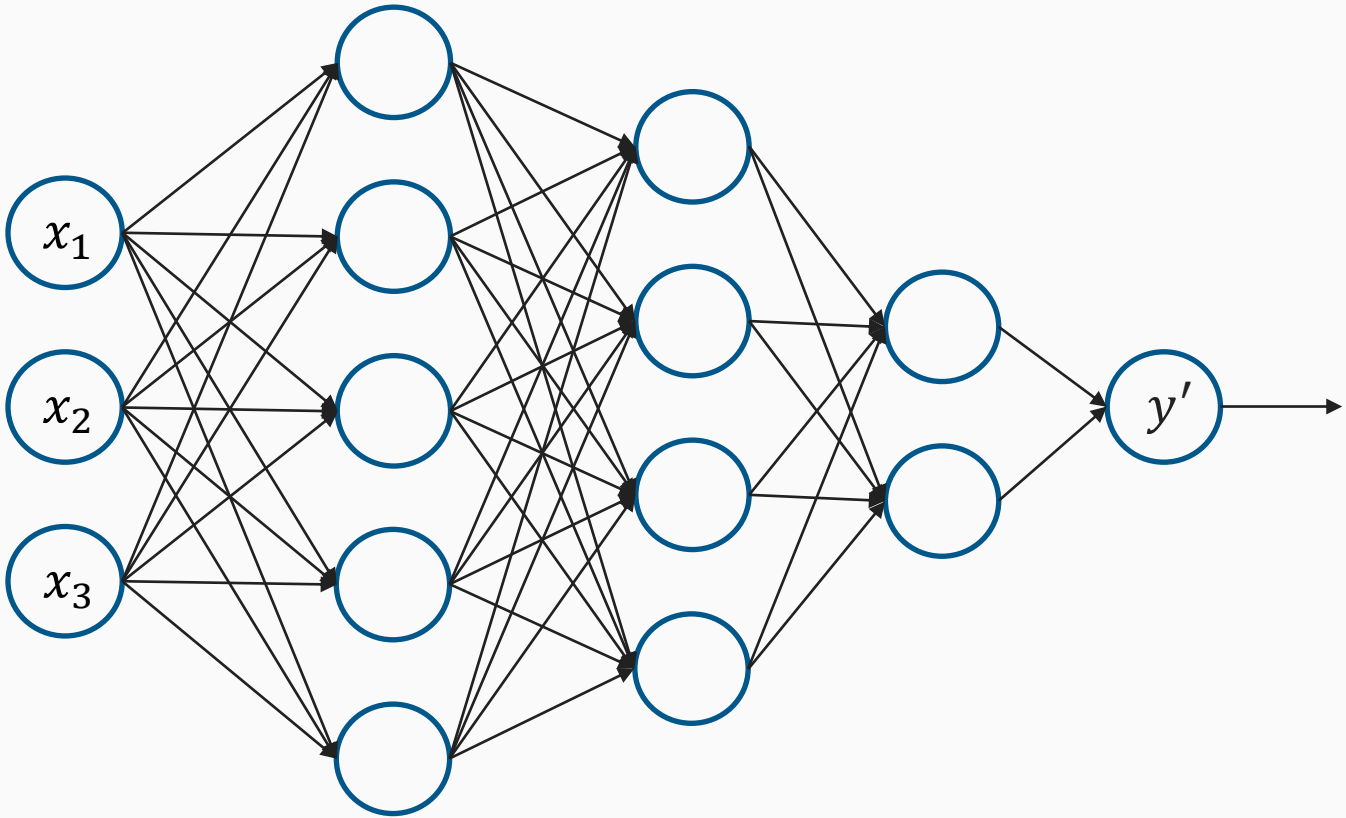
$$ReLU(x_1 w_{11} + x_2 w_{21} + b_1)$$

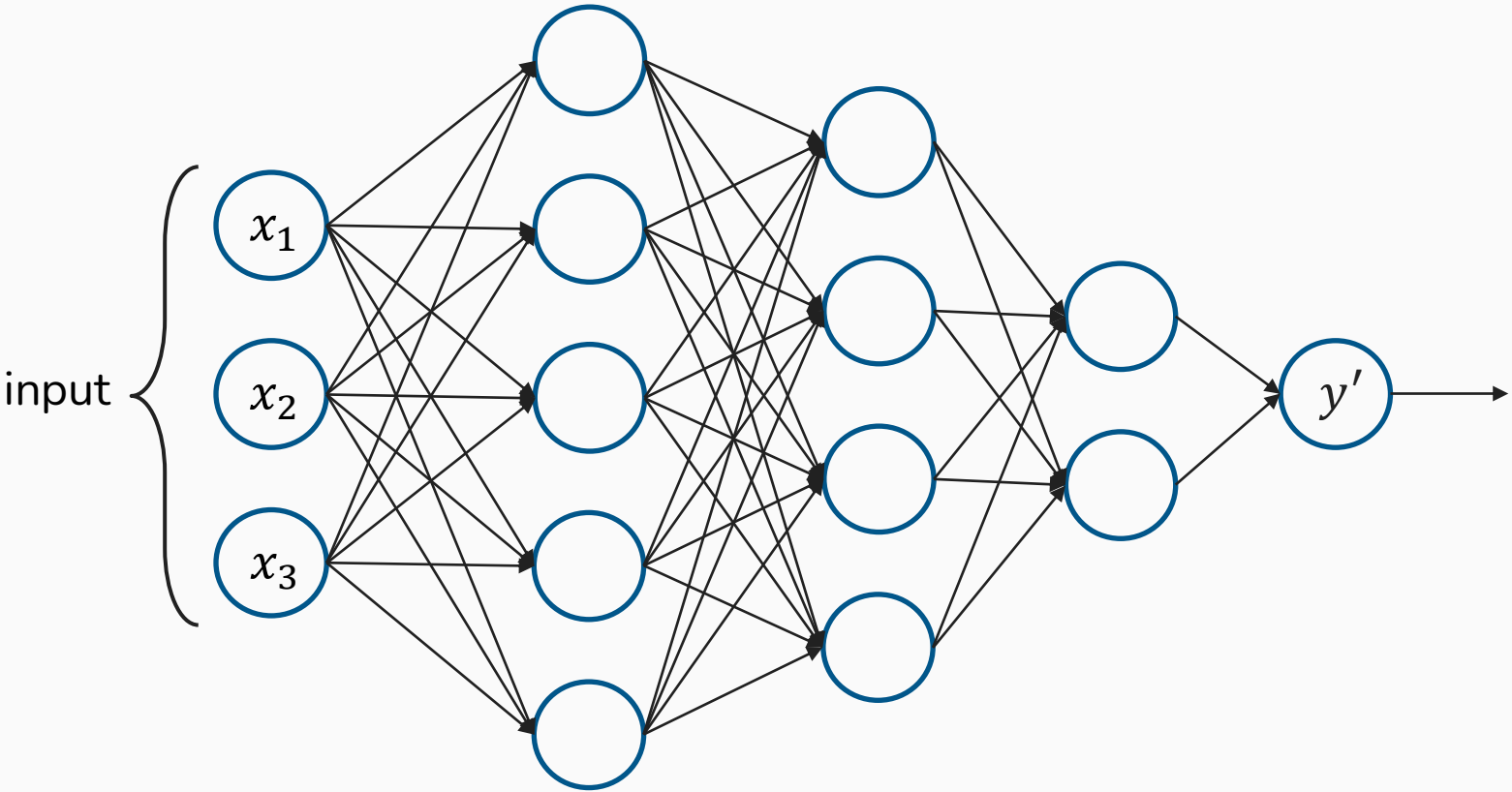
$$ReLU(x_1 w_{12} + x_2 w_{22} + b_2)$$

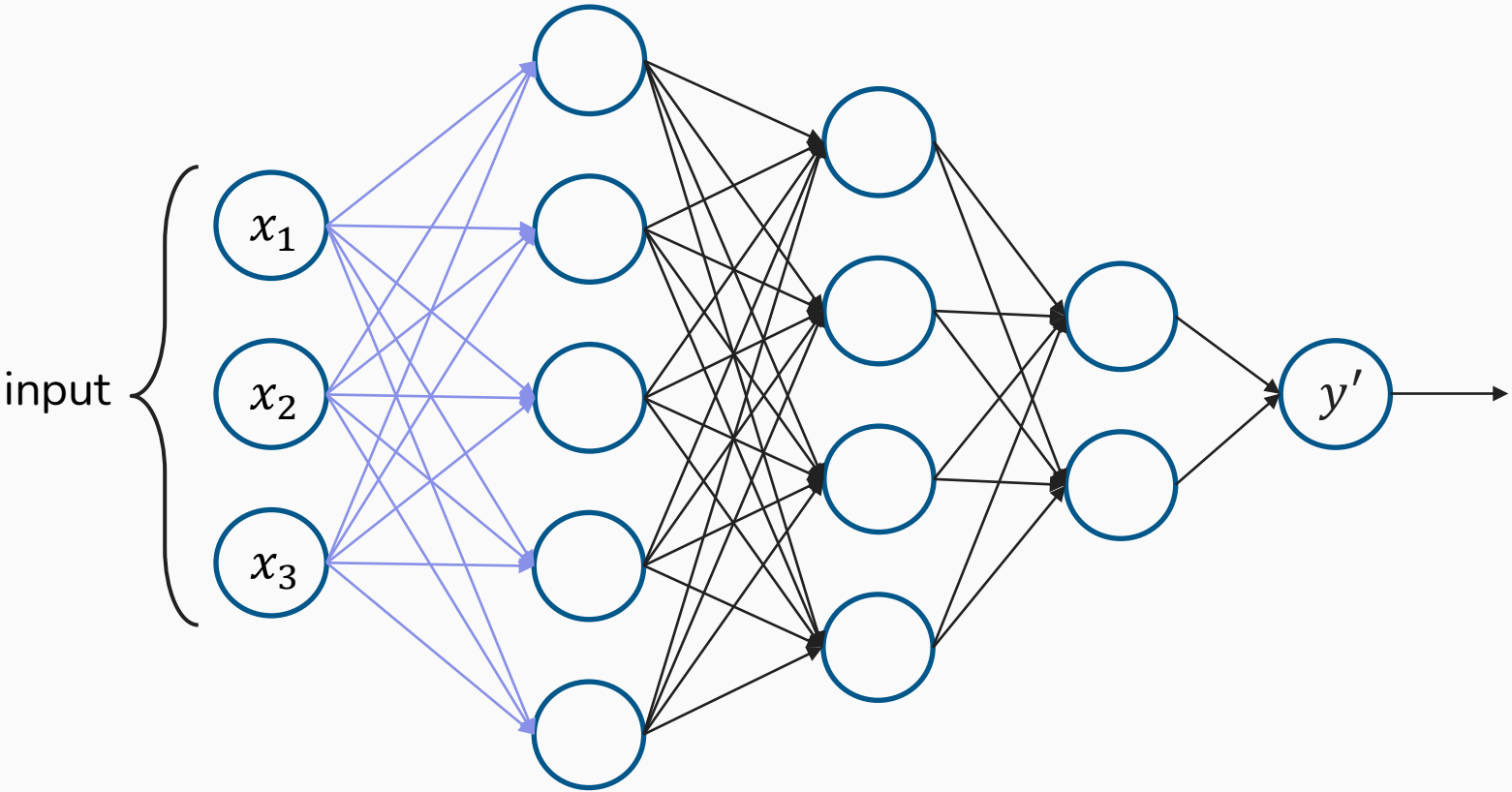
$$ReLU(x_1 w_{13} + x_2 w_{23} + b_3)$$

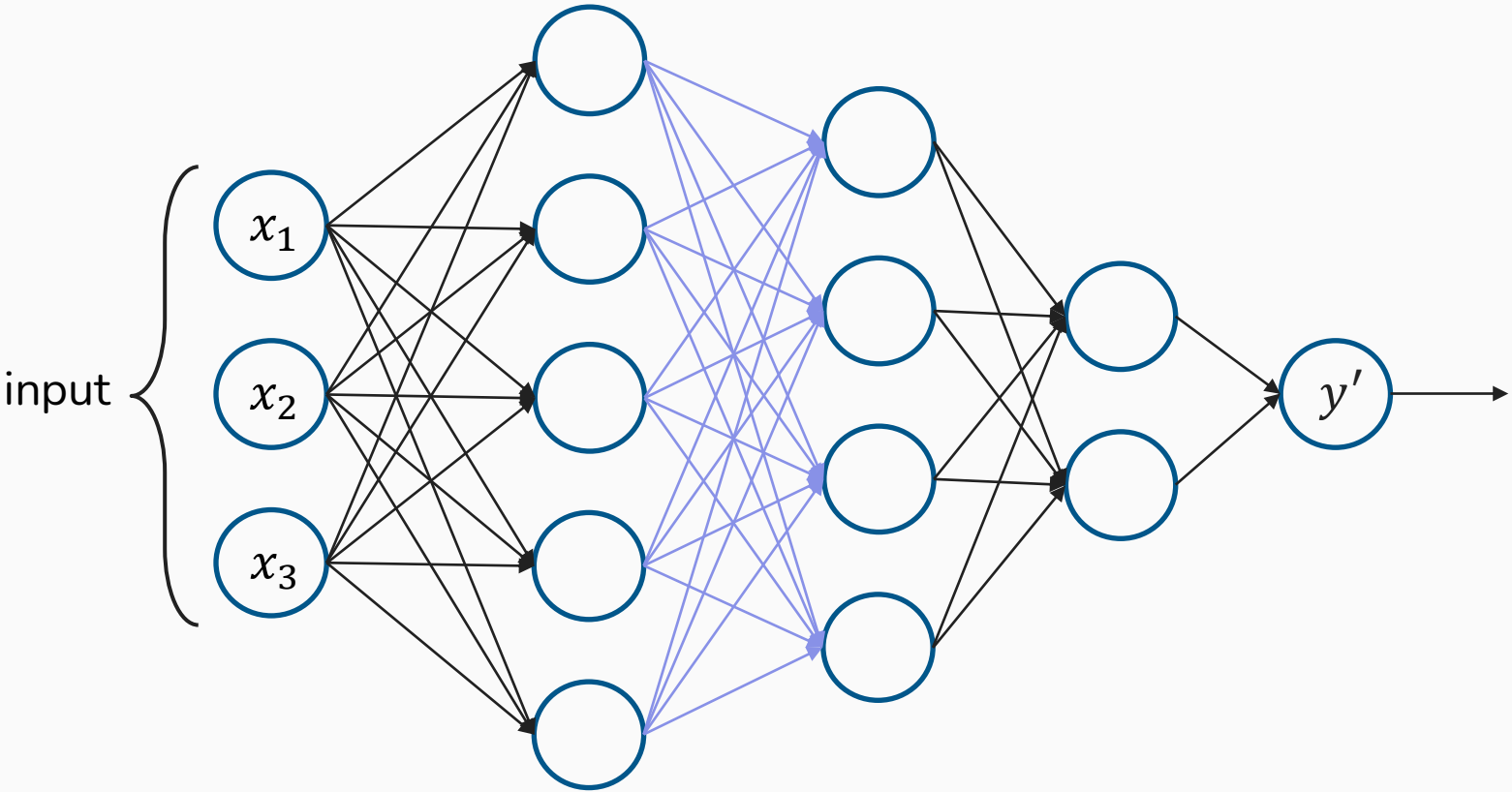
$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

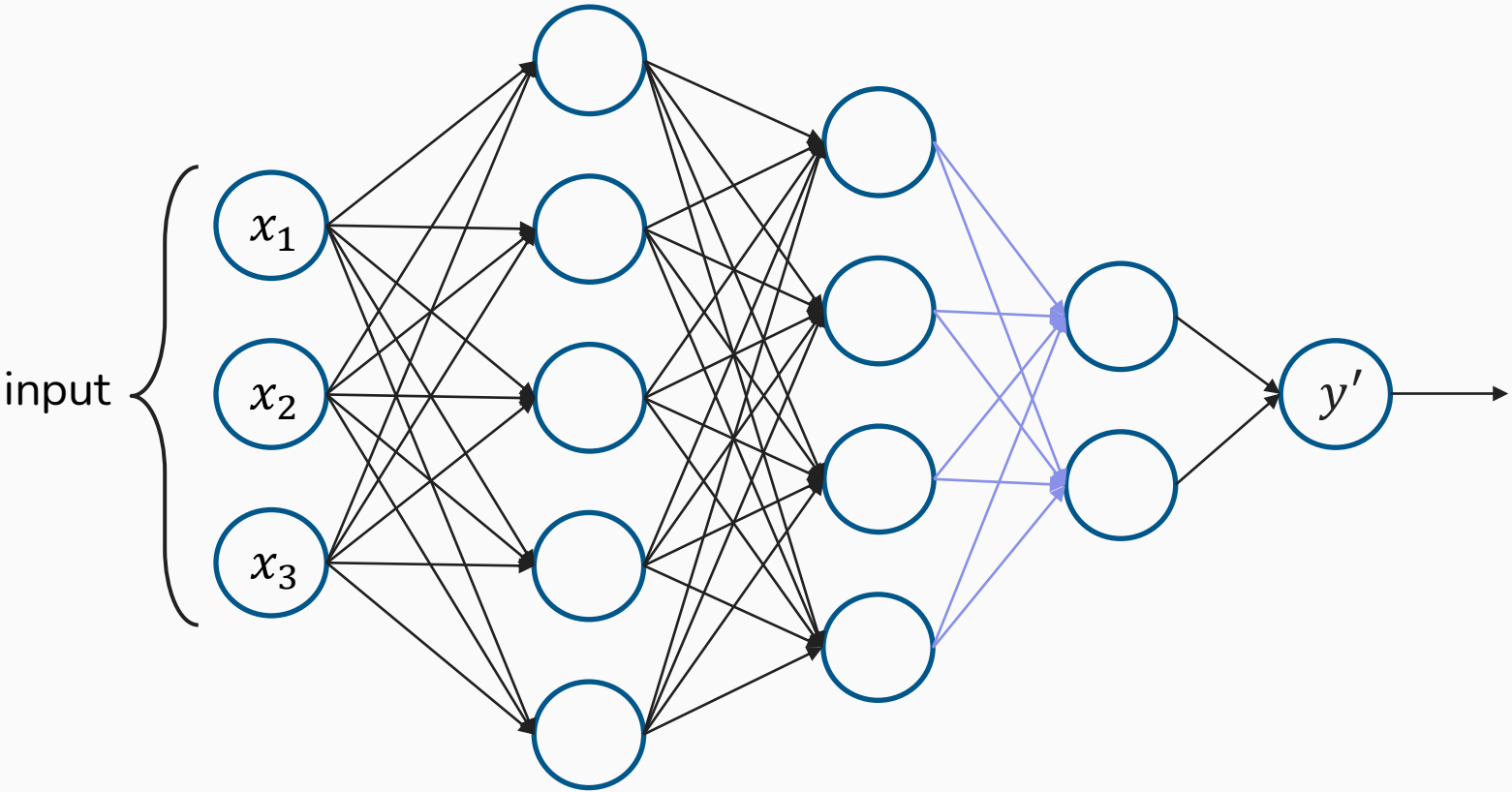

b_1
b_2
b_3
x_1
x_2
x_3
y'

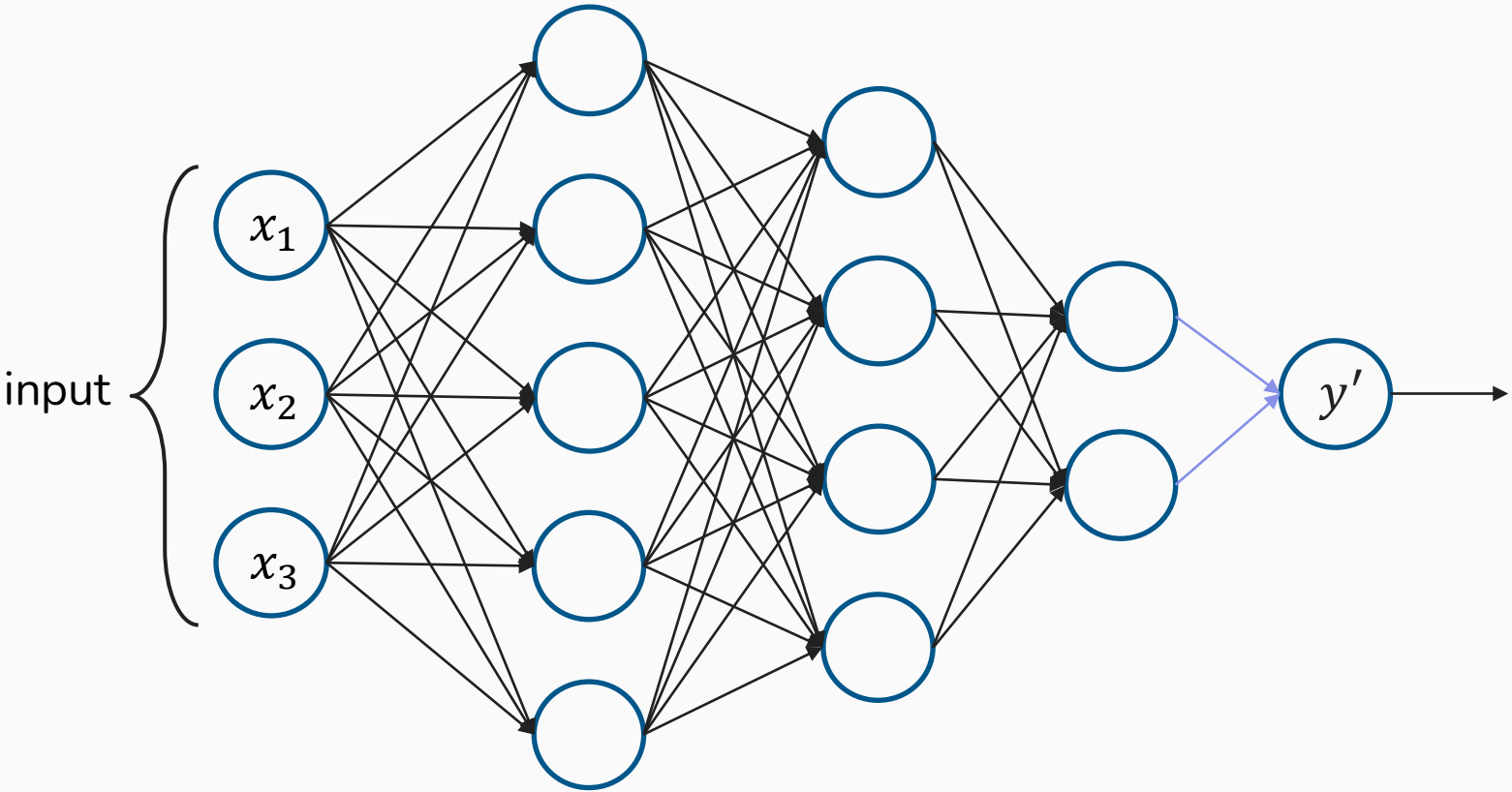

x_1
x_2
x_3
y'

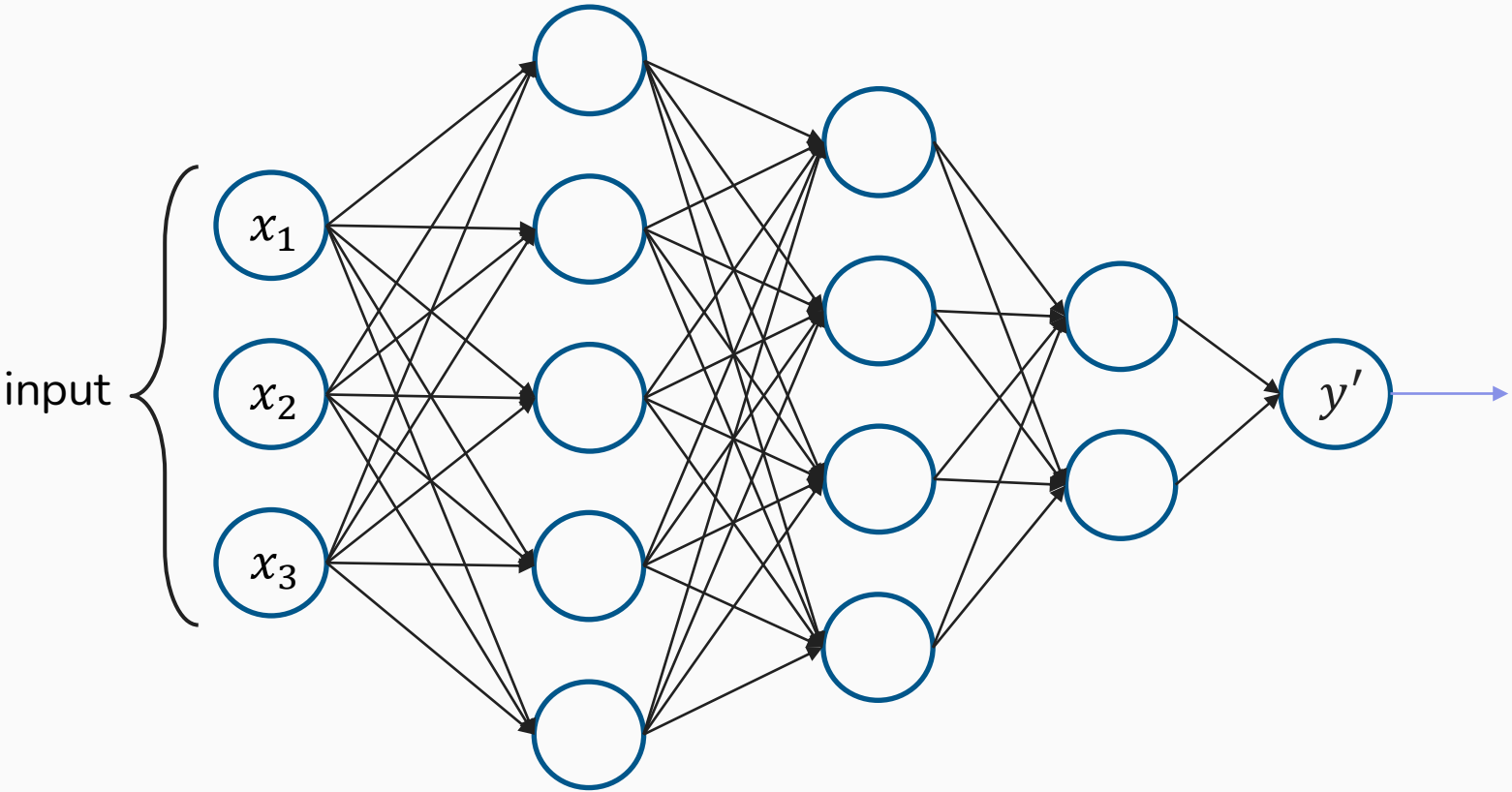

input
x_1
x_2
x_3
y'

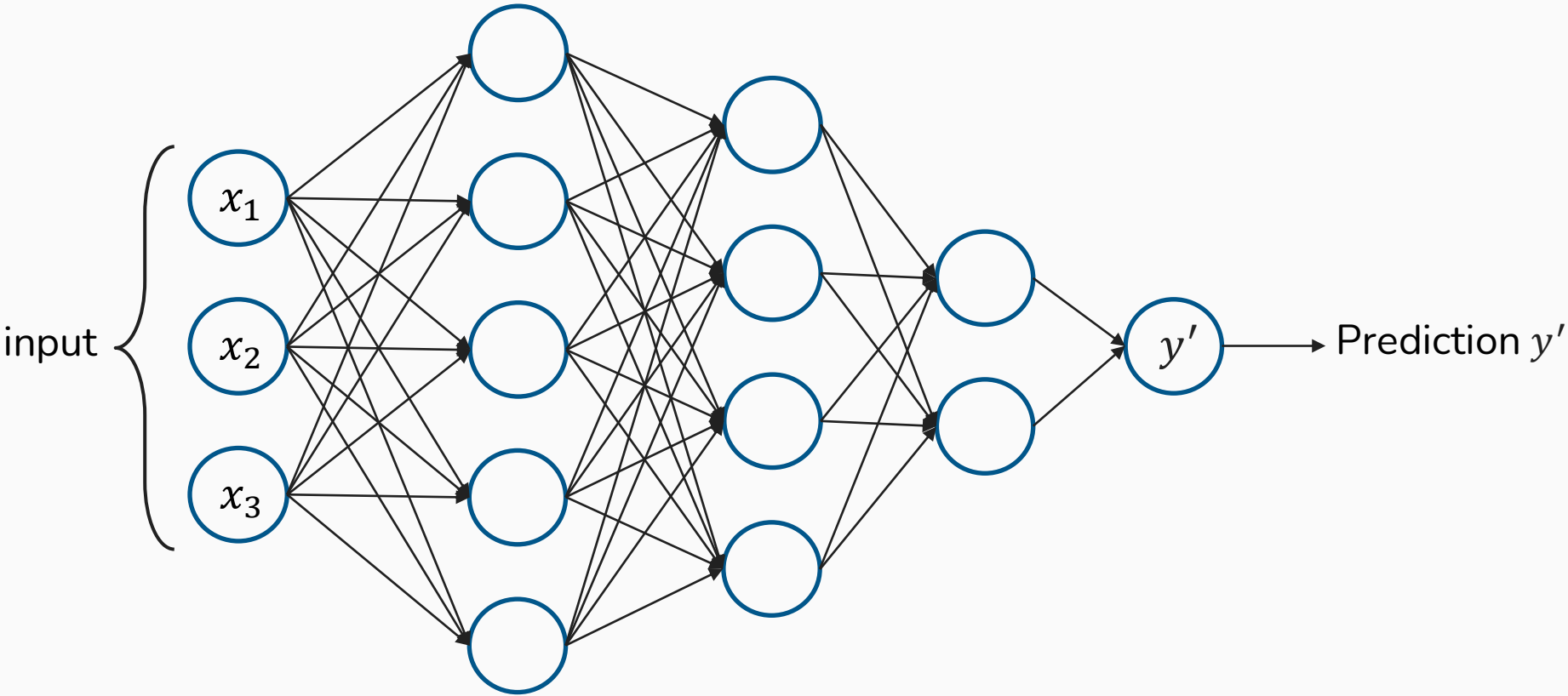

input
x_1
x_2
x_3
y'

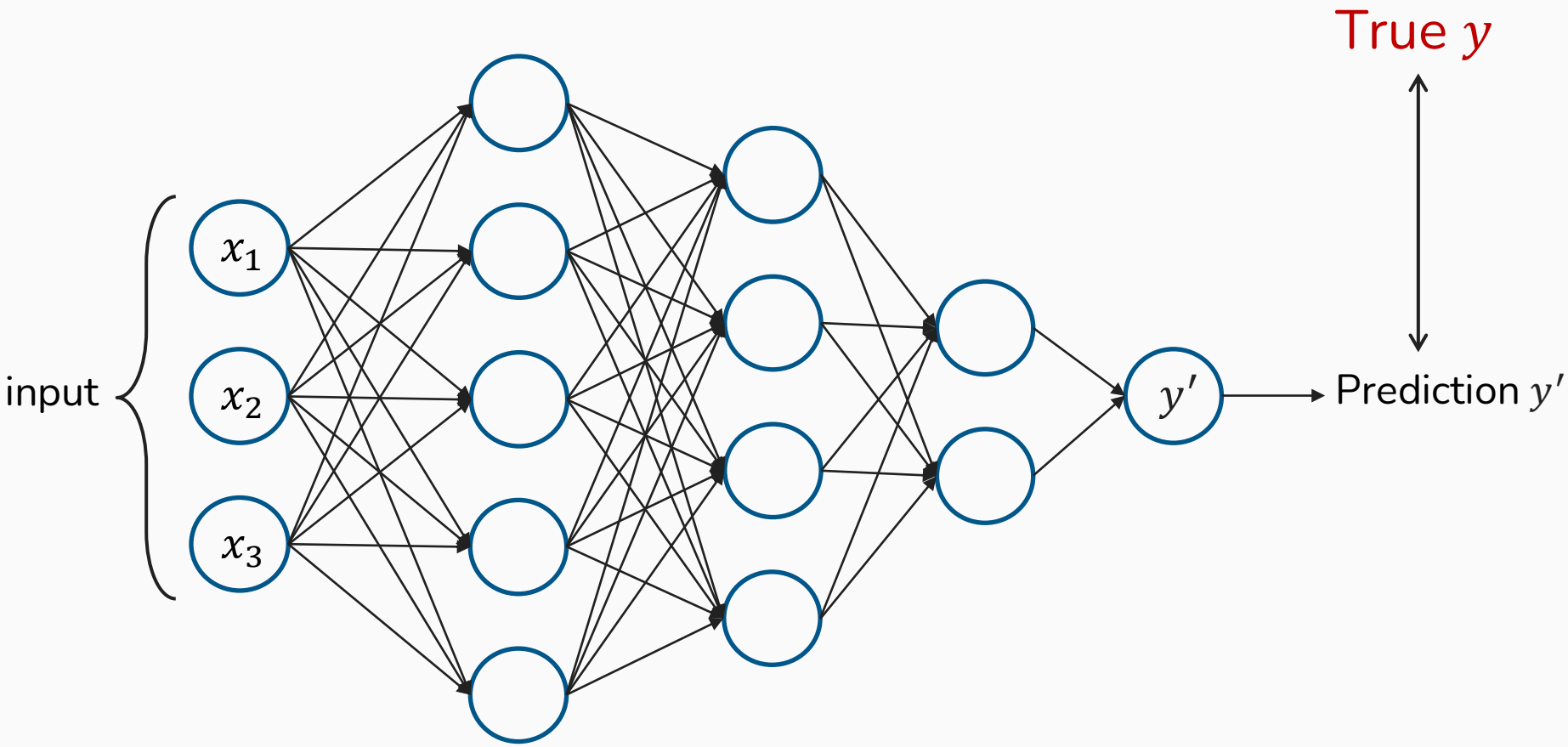

input
x_1
x_2
x_3
y'

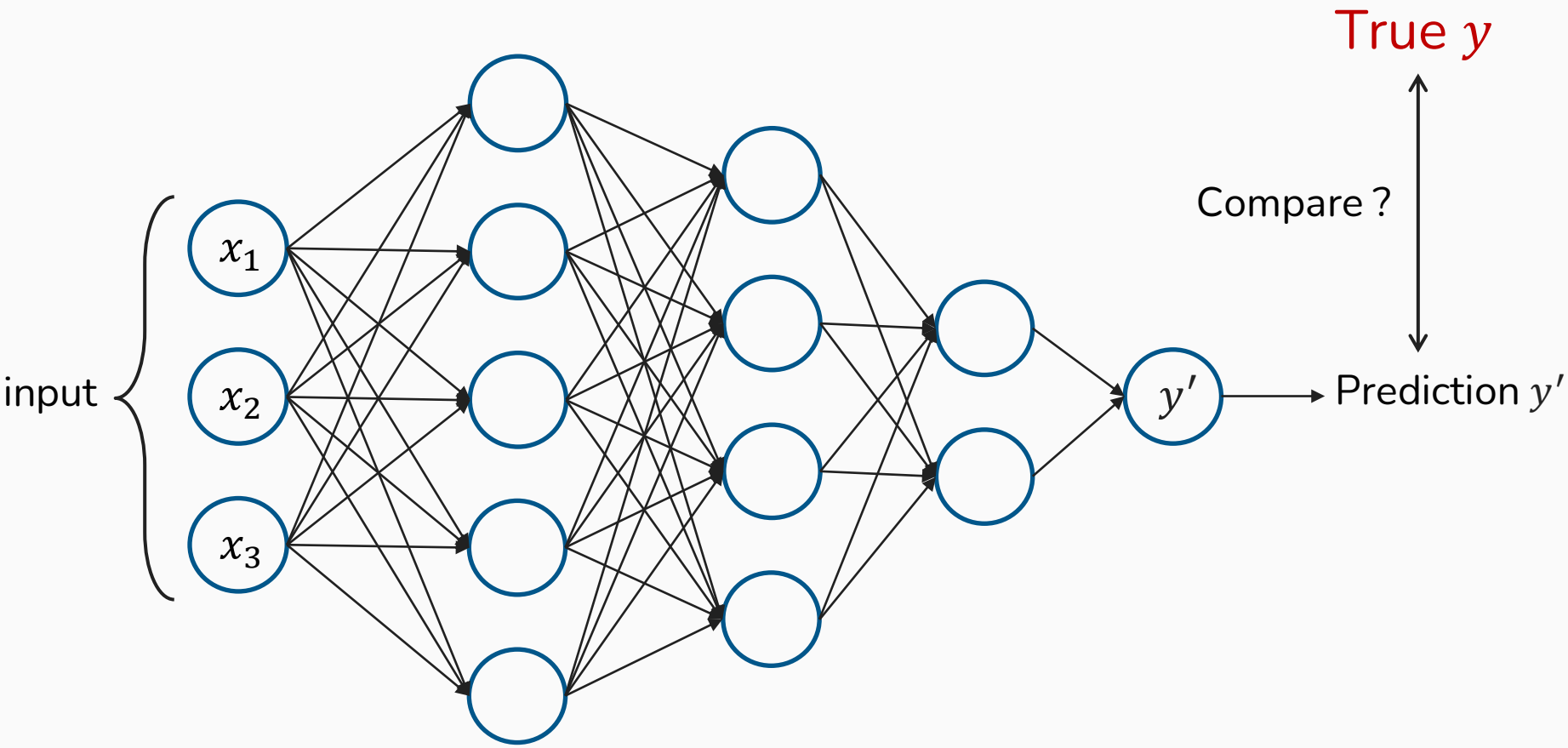

input
x_1
x_2
x_3
y'

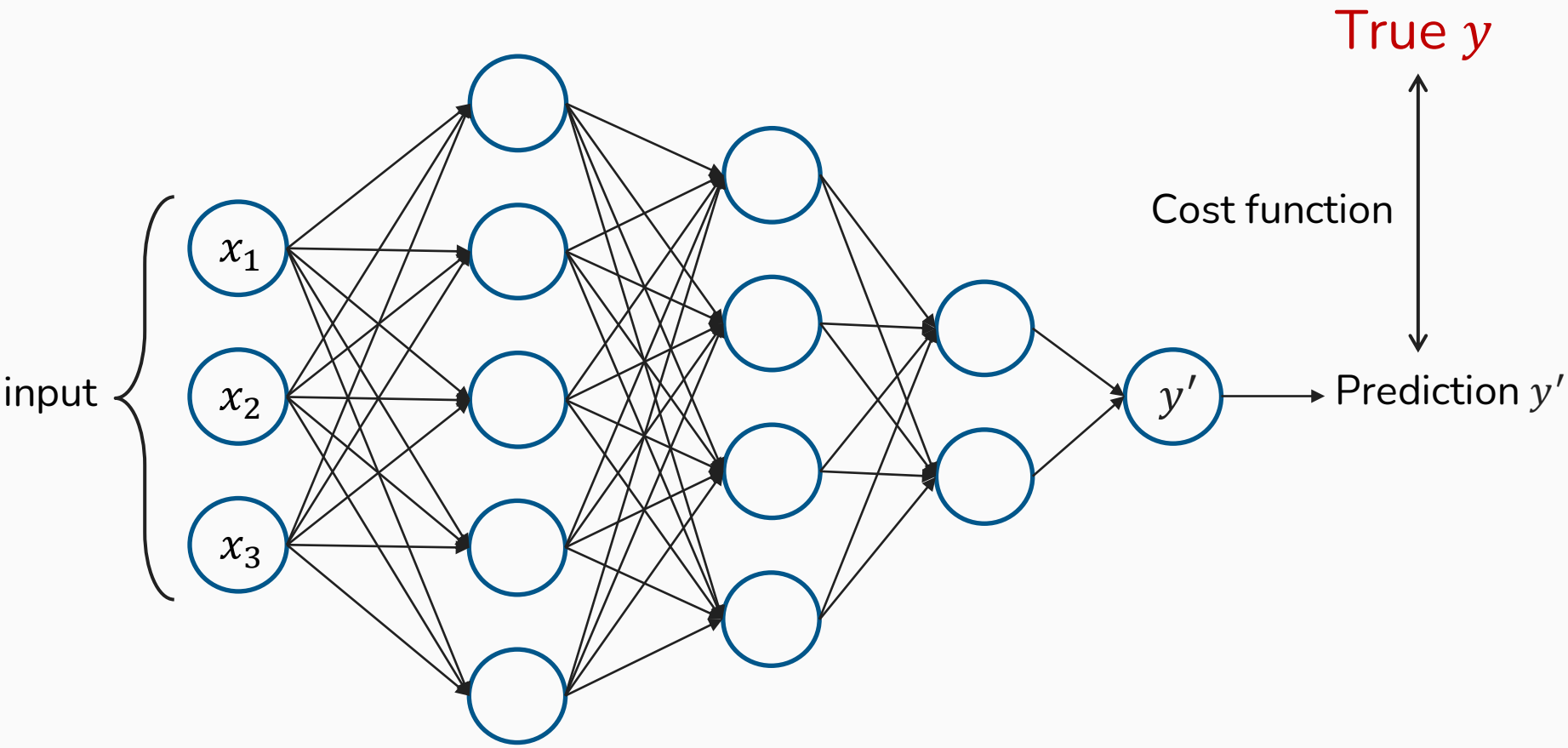

input
x_1
x_2
x_3
y'


input
x_1
x_2
x_3
y'


input
x_1
x_2
x_3
y'
Prediction y'


True y
input
x_1
x_2
x_3
y'
Prediction y'


True y
Compare ?
input
x_1
x_2
x_3
y'
Prediction y'


True y
Cost function
input
x_1
x_2
x_3
y'
Prediction y'

Cross Entropy

Cross Entropy

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(y'^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - y'^{(i)}\right)\right]$$

Cross Entropy

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(y'^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - y'^{(i)}\right)\right]$$

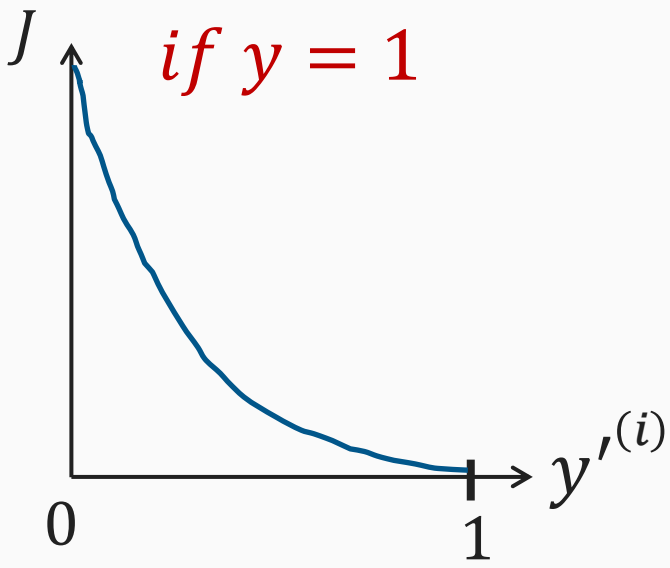
$$\Leftrightarrow$$

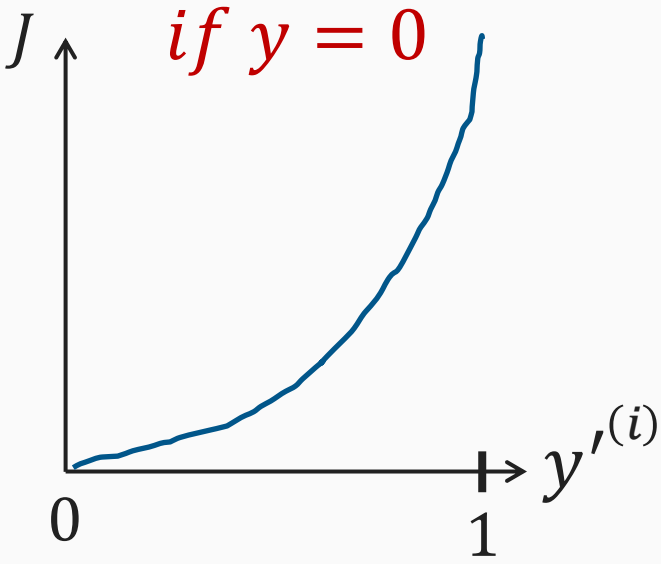
Cross Entropy

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(y'^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - y'^{(i)}\right)\right]$$

$$\Leftrightarrow$$

$$J = \begin{cases} -\log\left(y'^{(i)}\right) & if\ y = 1 \\ -\log\left(1 - y'^{(i)}\right) & if\ y = 0 \end{cases}$$



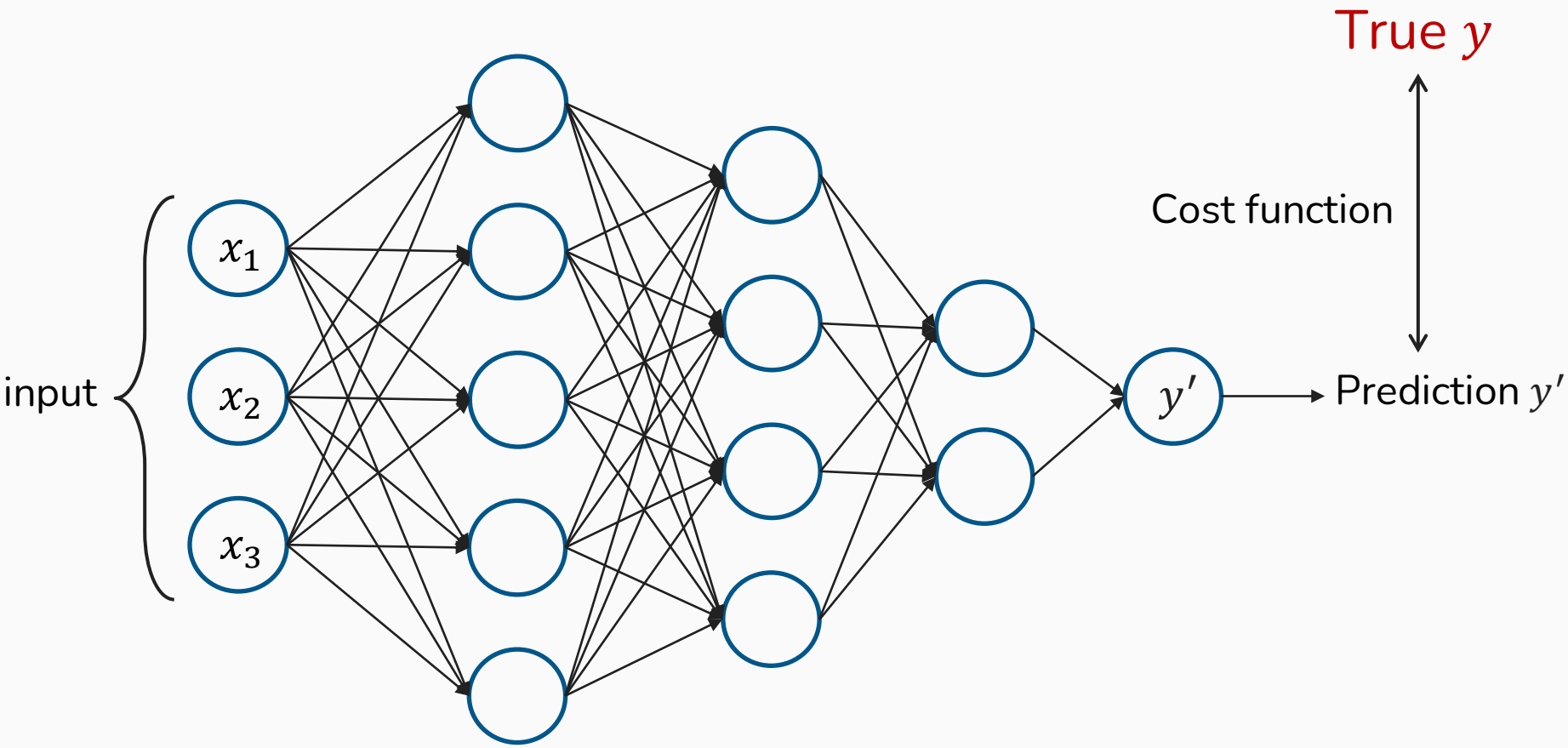


$$J = \begin{cases} -\log\left(y'^{(i)}\right) & if\ y = 1 \\ -\log\left(1 - y'^{(i)}\right) & if\ y = 0 \end{cases}$$

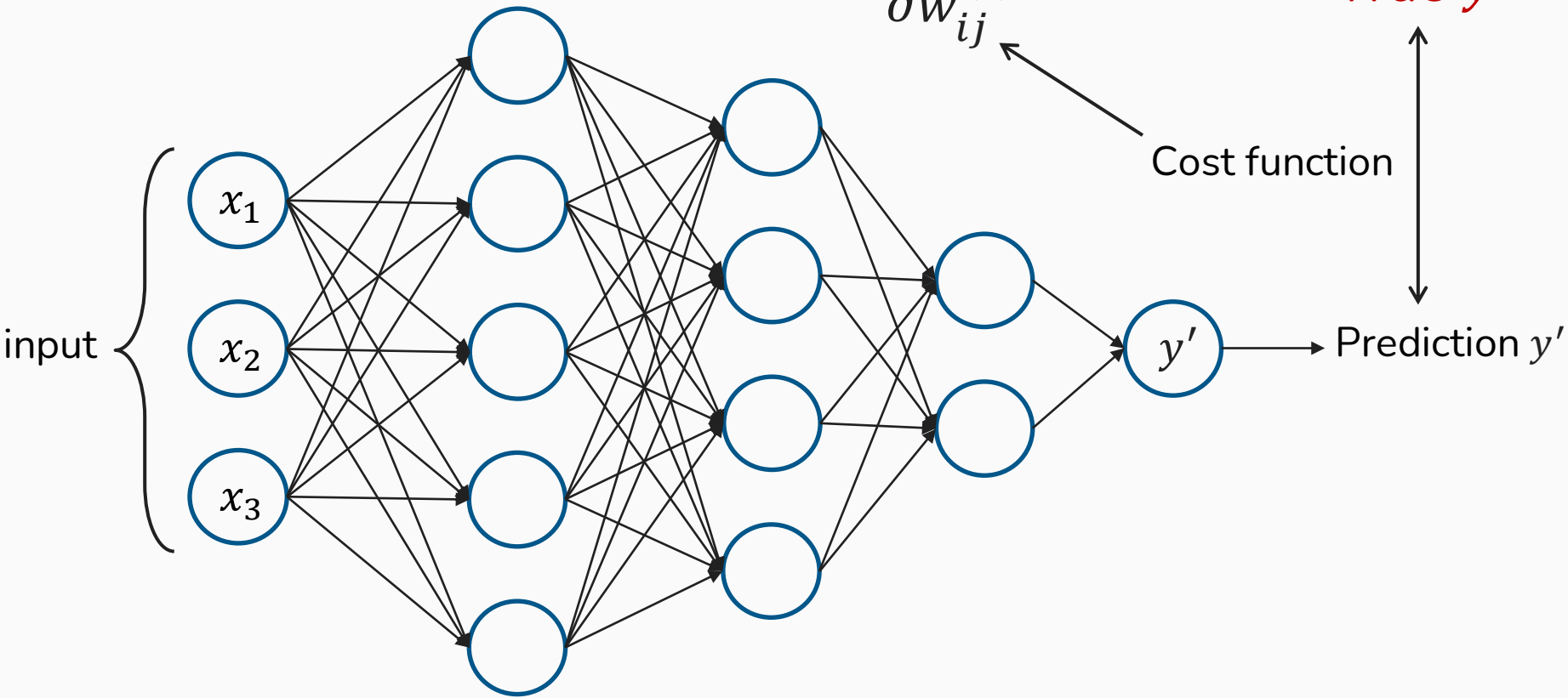
Mean squared error

Mean squared error

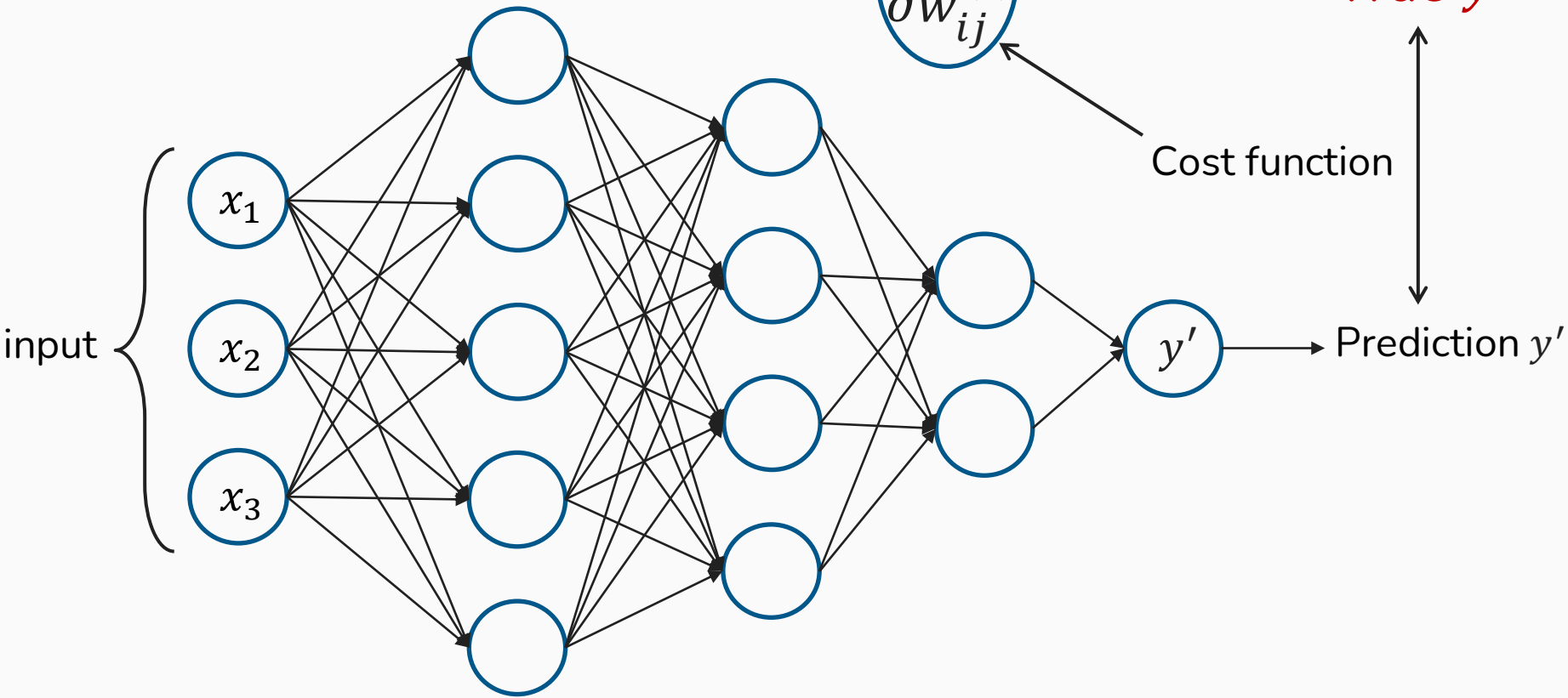
$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y'^{(i)} - y^{(i)}\right)^2$$


True y
Cost function
input
x_1
x_2
x_3
y'
Prediction y'

$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$



$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$

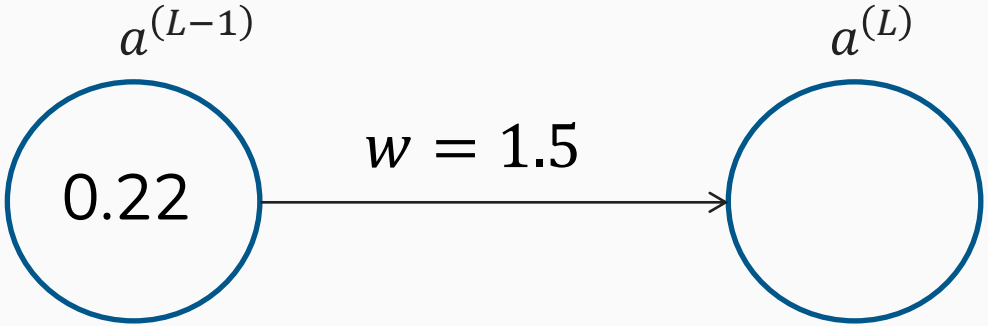


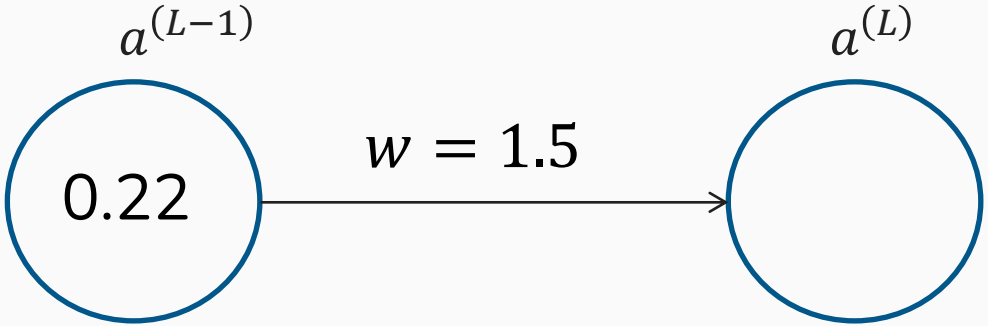
backpropagation

$a^{(L)}$

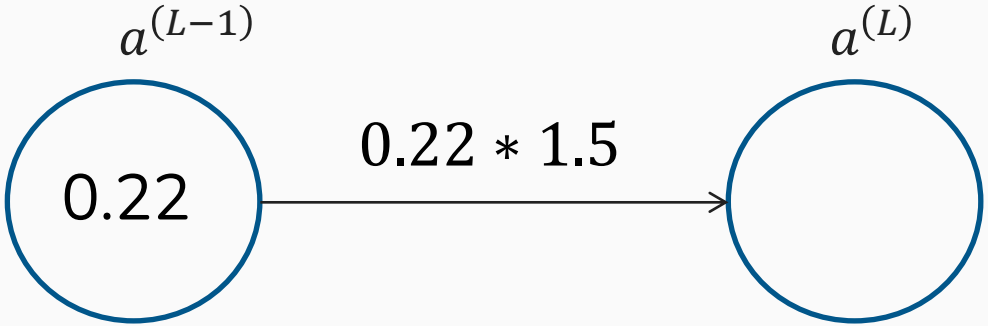
$a^{(L-1)}$
$a^{(L)}$

$a^{(L-1)}$
$a^{(L)}$
0.22


$a^{(L-1)}$
$a^{(L)}$
0.22
$w = 1.5$



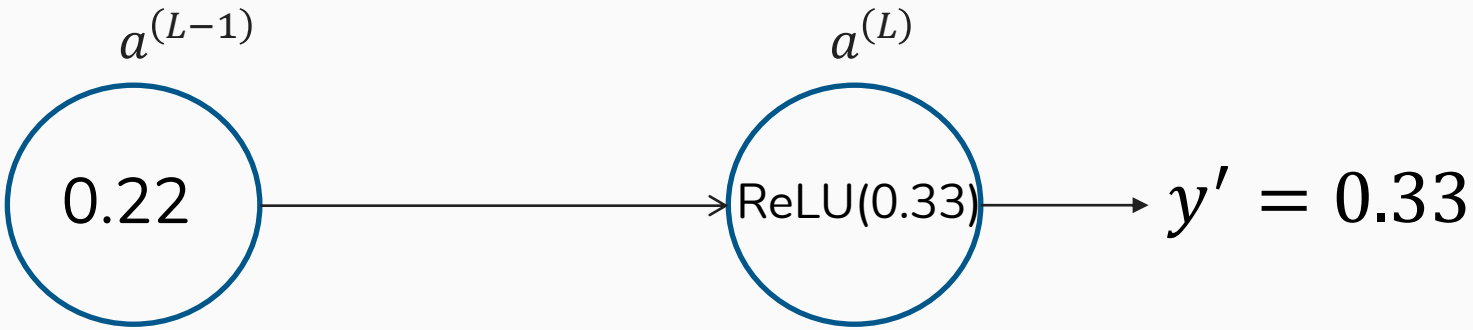
$y = 1$

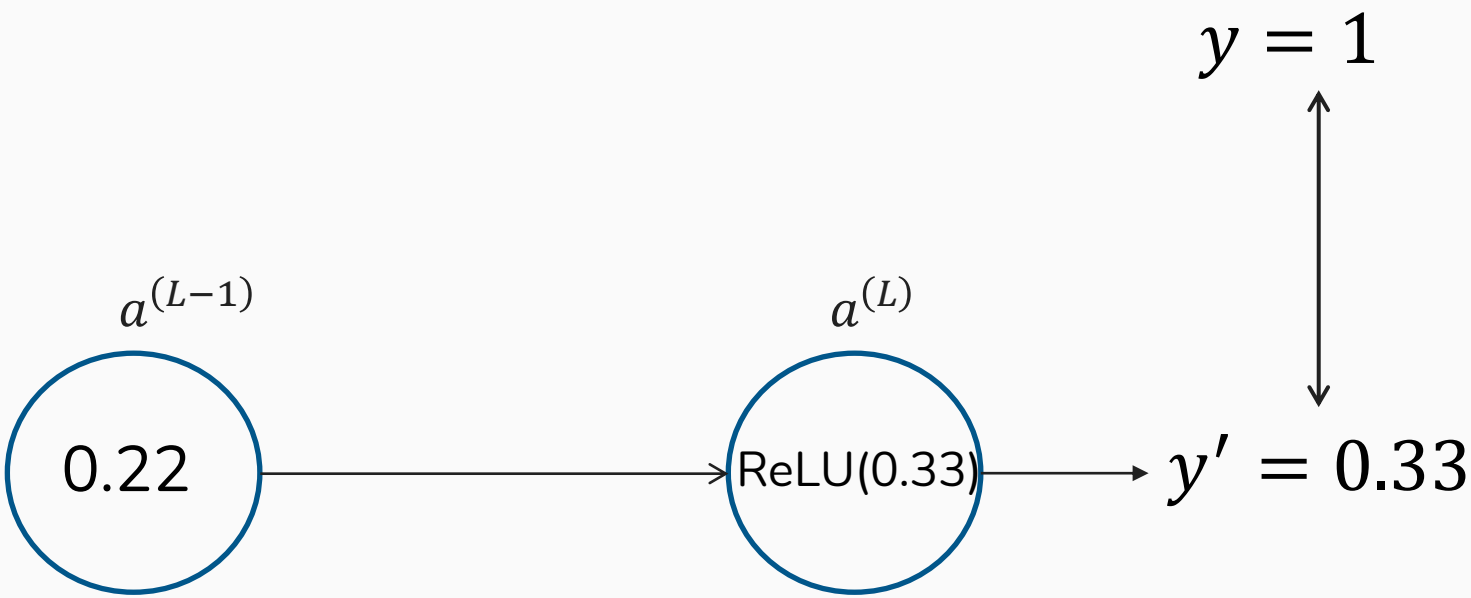


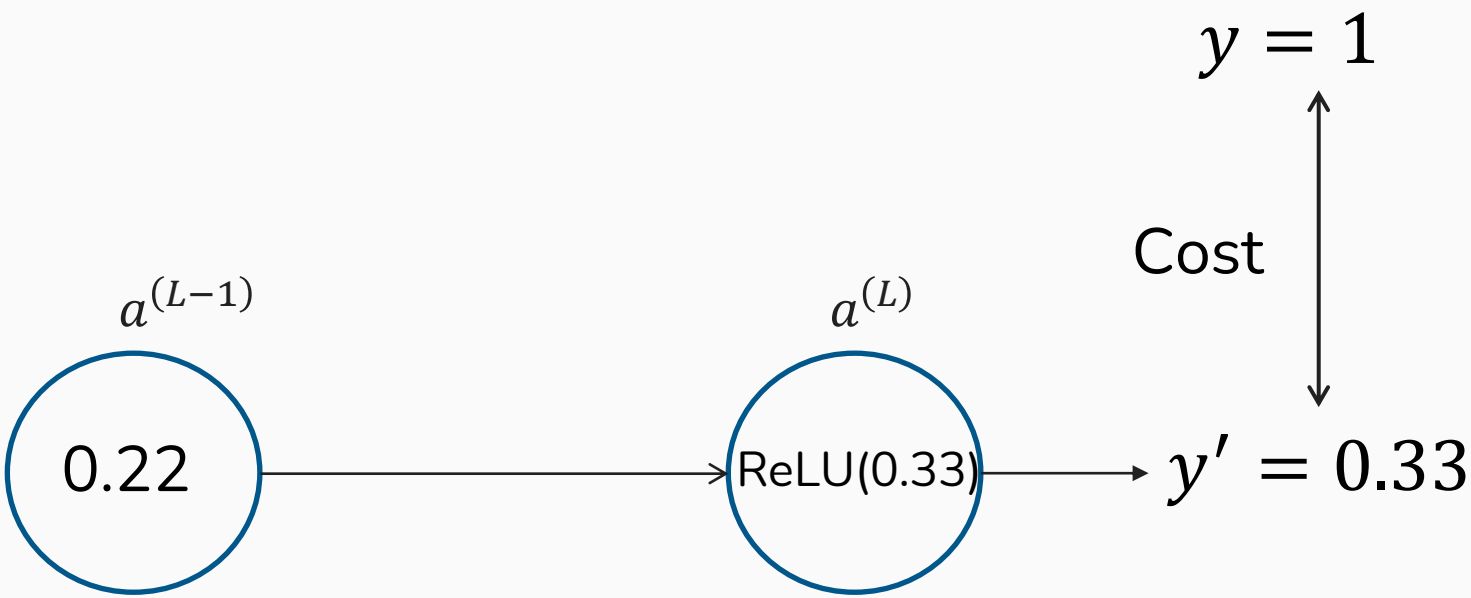
$y = 1$

$y = 1$

$y = 1$




$y = 1$
$a^{(L-1)}$
$a^{(L)}$
0.22
ReLU(0.33)
$y' = 0.33$


$y = 1$
Cost
$a^{(L-1)}$
$a^{(L)}$
0.22
ReLU(0.33)
$y' = 0.33$

$$a^{(L-1)}$$

$$w^{(L)}$$

$$a^{(L-1)}$$

$$w^{(L)}$$

$$a^{(L-1)} \rightarrow$$

$$w^{(L)} \rightarrow$$

$$w^{(L)}$$

$$a^{(L-1)} \rightarrow$$

$$w^{(L)} \rightarrow$$

$$w^{(L)} * a^{(L-1)}$$

$$a^{(L-1)} \rightarrow$$

$$w^{(L)} \rightarrow$$

$$z^{(L)} = w^{(L)} * a^{(L-1)}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \longrightarrow$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \longrightarrow a^{(L)}$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$a^{(L-1)} \searrow$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \longrightarrow a^{(L)} \longrightarrow$$

$$w^{(L)} \nearrow$$

$$y$$

$$a^{(L-1)} \rightarrow$$

$$w^{(L)} \rightarrow$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \longrightarrow a^{(L)} \longrightarrow$$

$$a^{(L-1)} \rightarrow z^{(L)}$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)}$$

$$y \rightarrow$$

$$a^{(L)} \rightarrow$$

$$
\begin{tikzcd}
 & & & y \arrow[dr] & \\
a^{(L-1)} \arrow[dr] & & & & \\
 & z^{(L)} = w^{(L)} * a^{(L-1)} \arrow[r] & a^{(L)} \arrow[r] & & J \\
w^{(L)} \arrow[ur] & & & &
\end{tikzcd}
$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:

$$\frac{\partial J}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:

$$\frac{\partial J}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:

$$\frac{\partial J}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:

$$\frac{\partial J}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:

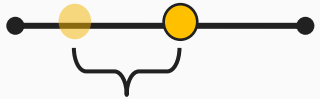
$$\frac{\partial J}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:



$$\frac{\partial J}{\partial w^{(L)}}$$

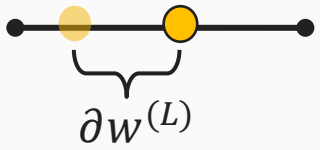
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:

$$\frac{\partial J}{\partial w^{(L)}}$$

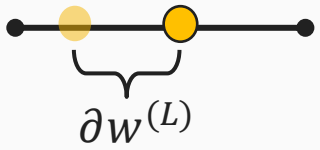


$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:

$$\frac{\partial J}{\partial w^{(L)}}$$

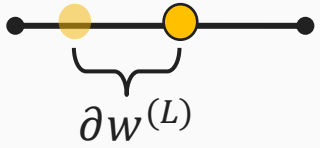
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:

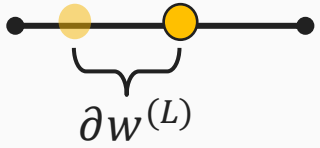
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:

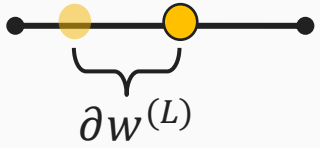
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:

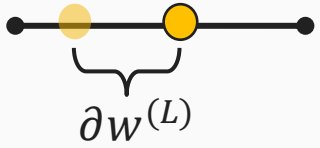
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:

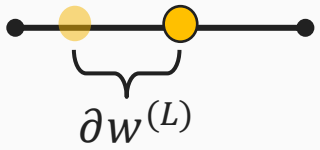
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

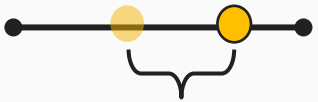
$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:



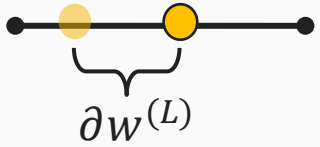
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

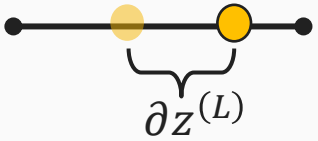
$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

A small change in $w^{(L)}$:



Causes a change in $z^{(L)}$:



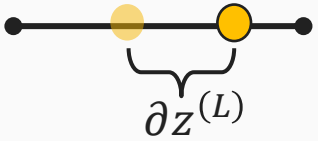
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $z^{(L)}$:



Causes a change in $a^{(L)}$:

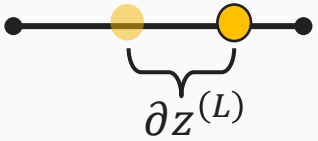
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $z^{(L)}$:



Causes a change in $a^{(L)}$:

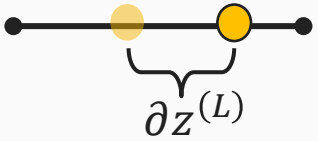
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $z^{(L)}$:



Causes a change in $a^{(L)}$:

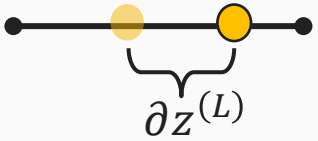
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $z^{(L)}$:



Causes a change in $a^{(L)}$:

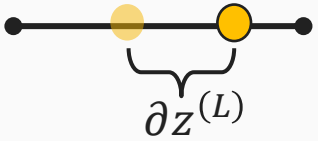
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

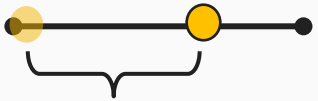
$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $z^{(L)}$:



Causes a change in $a^{(L)}$:



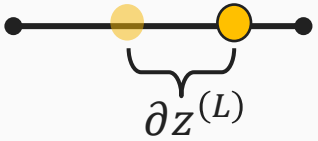
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

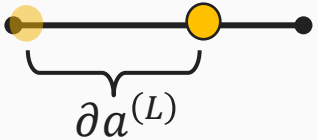
$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $z^{(L)}$:



Causes a change in $a^{(L)}$:



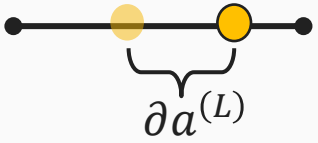
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $a^{(L)}$:



Causes a change in J:

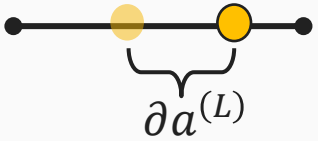
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $a^{(L)}$:



Causes a change in J:

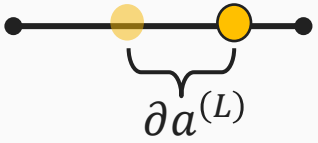
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $a^{(L)}$:



Causes a change in J:

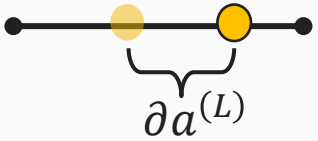
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $a^{(L)}$:



Causes a change in J:

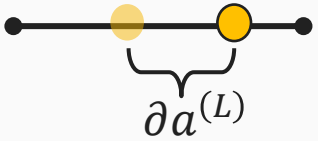
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $a^{(L)}$:



Causes a change in J:

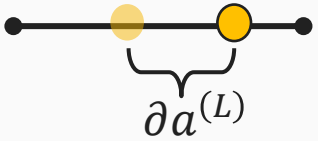
$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

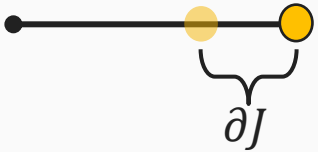
$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

And a small change in $a^{(L)}$:



Causes a change in J:



$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}}$$

$$
\begin{tikzcd}
 & & & y \arrow[dr] & \\
a^{(L-1)} \arrow[dr] & & & & \\
 & z^{(L)} = w^{(L)} * a^{(L-1)} \arrow[r] & & a^{(L)} \arrow[r] & J \\
w^{(L)} \arrow[ur] & & & &
\end{tikzcd}
$$

$$\frac{\partial J}{\partial w^{(L)}} =$$

$$a^{(L-1)} \rightarrow z^{(L)}, \quad w^{(L)} \rightarrow z^{(L)}$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J, \quad y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}} =$$

$$a^{(L-1)} \rightarrow z^{(L)}$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}} =$$

$$a^{(L-1)} \rightarrow z^{(L)}, \quad w^{(L)} \rightarrow z^{(L)}, \quad y \rightarrow J$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

$$
\begin{tikzcd}
a^{(L-1)} \arrow[dr] & & & y \arrow[dr] & \\
& z^{(L)} = w^{(L)} * a^{(L-1)} \arrow[r] & a^{(L)} \arrow[r] & & J \\
w^{(L)} \arrow[ur] & & & &
\end{tikzcd}
$$

$$
\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}
$$

$$a^{(L-1)} \rightarrow z^{(L)}, \quad w^{(L)} \rightarrow z^{(L)}, \quad y \rightarrow J$$

$$z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J$$

$$w^{(L)} \rightarrow z^{(L)}$$

$$y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$$

$$a^{(L-1)} \rightarrow z^{(L)}, \quad w^{(L)} \rightarrow z^{(L)}, \quad z^{(L)} = w^{(L)} * a^{(L-1)} \rightarrow a^{(L)} \rightarrow J, \quad y \rightarrow J$$

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$$

$$
\begin{tikzcd}
a^{(L-1)} \arrow[dr] & & & y \arrow[dr] & \\
& z^{(L)} = w^{(L)} * a^{(L-1)} \arrow[r] & a^{(L)} \arrow[r] & & J \\
w^{(L)} \arrow[ur] & & & &
\end{tikzcd}
$$

$$
\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial J}{\partial a^{(L)}}
$$

$$\begin{tikzcd}
 & & & y \arrow[dr] & \\
a^{(L-1)} \arrow[dr] & & & & \\
 & z^{(L)} = w^{(L)} * a^{(L-1)} \arrow[r] & & a^{(L)} \arrow[r] & J \\
w^{(L)} \arrow[ur] & & & &
\end{tikzcd}$$

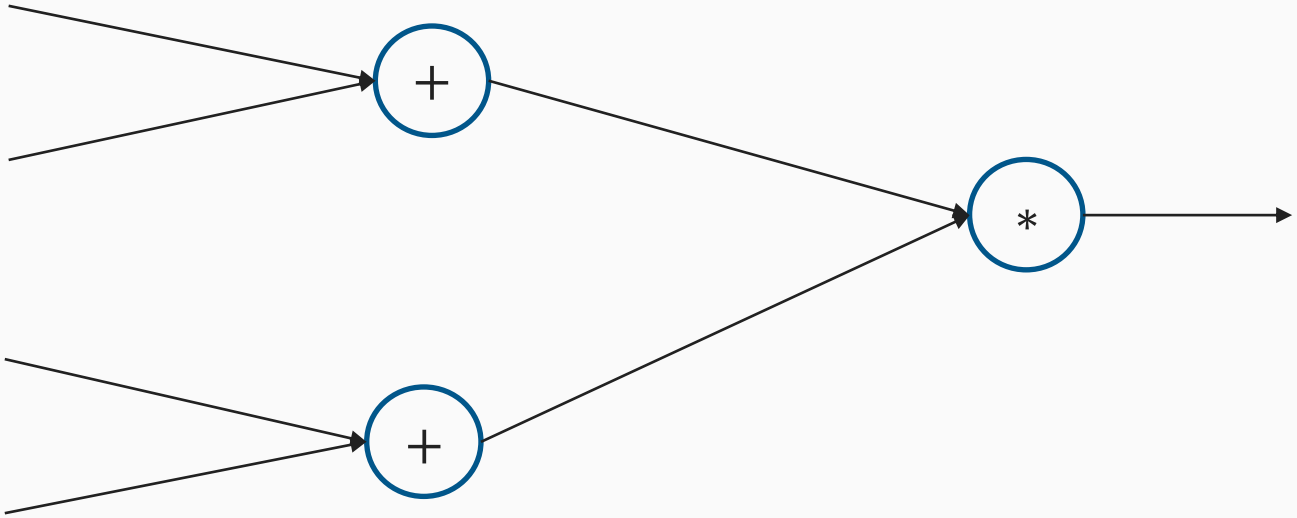
Chain rule:

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial J}{\partial a^{(L)}}$$

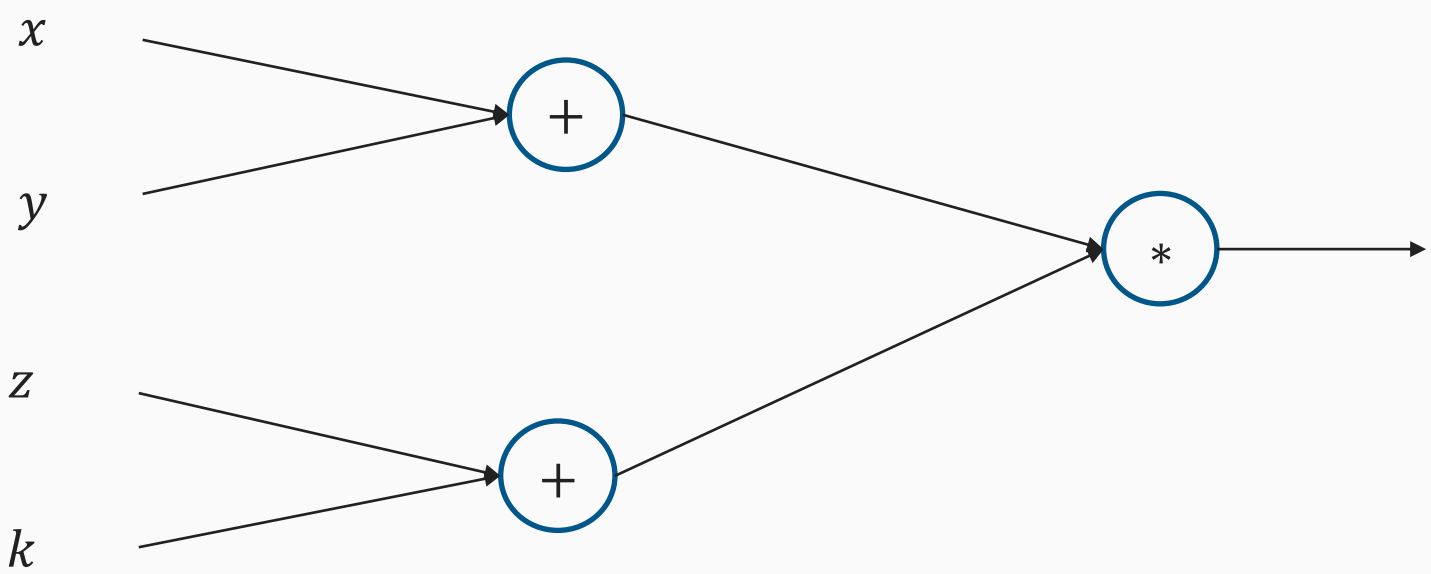
Back-propagation is one particular way to compute the chain rule formula. It is an efficient strategy that avoids redoing some of the same computations over and over again.

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

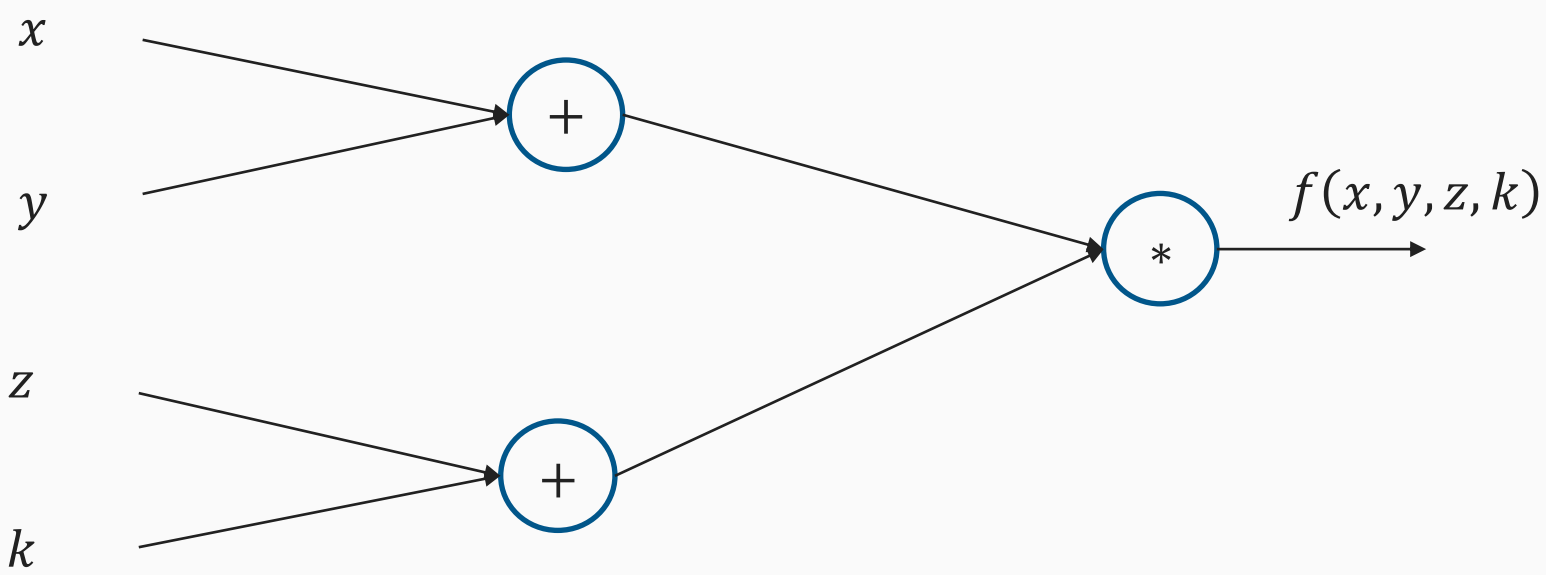
$$f(x, y, z, k) = (x + y)(z + k)$$



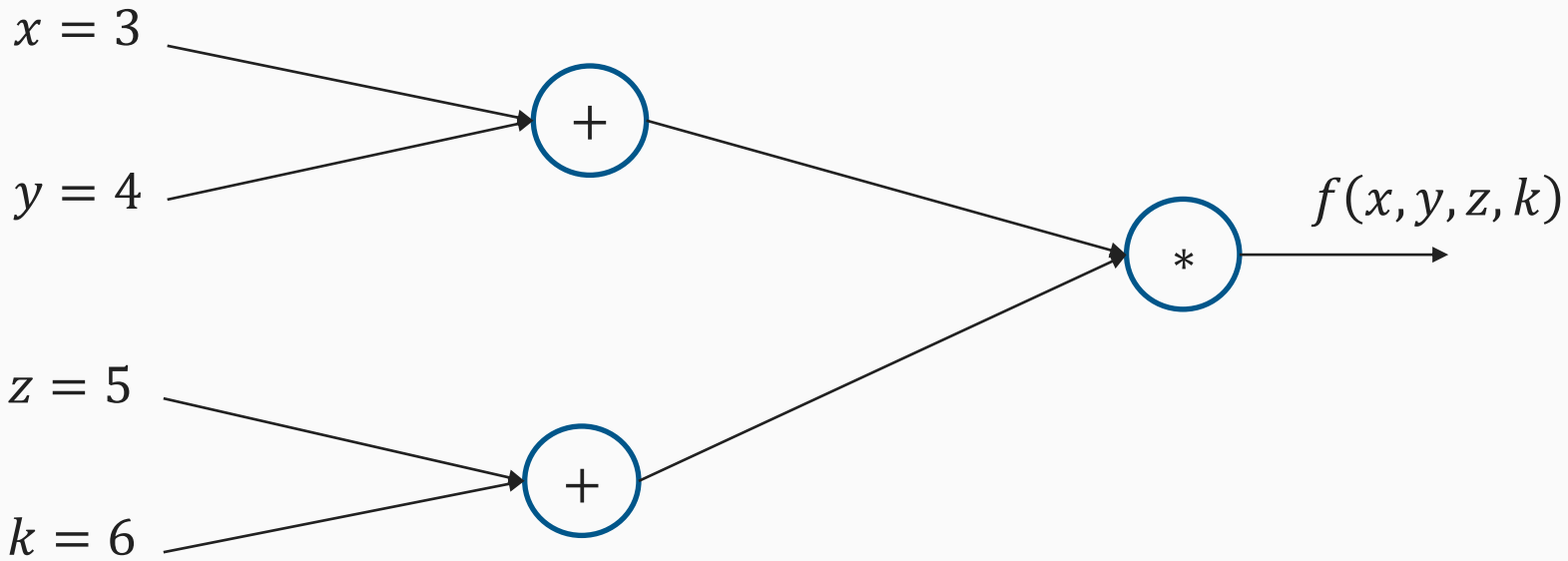
$$f(x, y, z, k) = (x + y)(z + k)$$



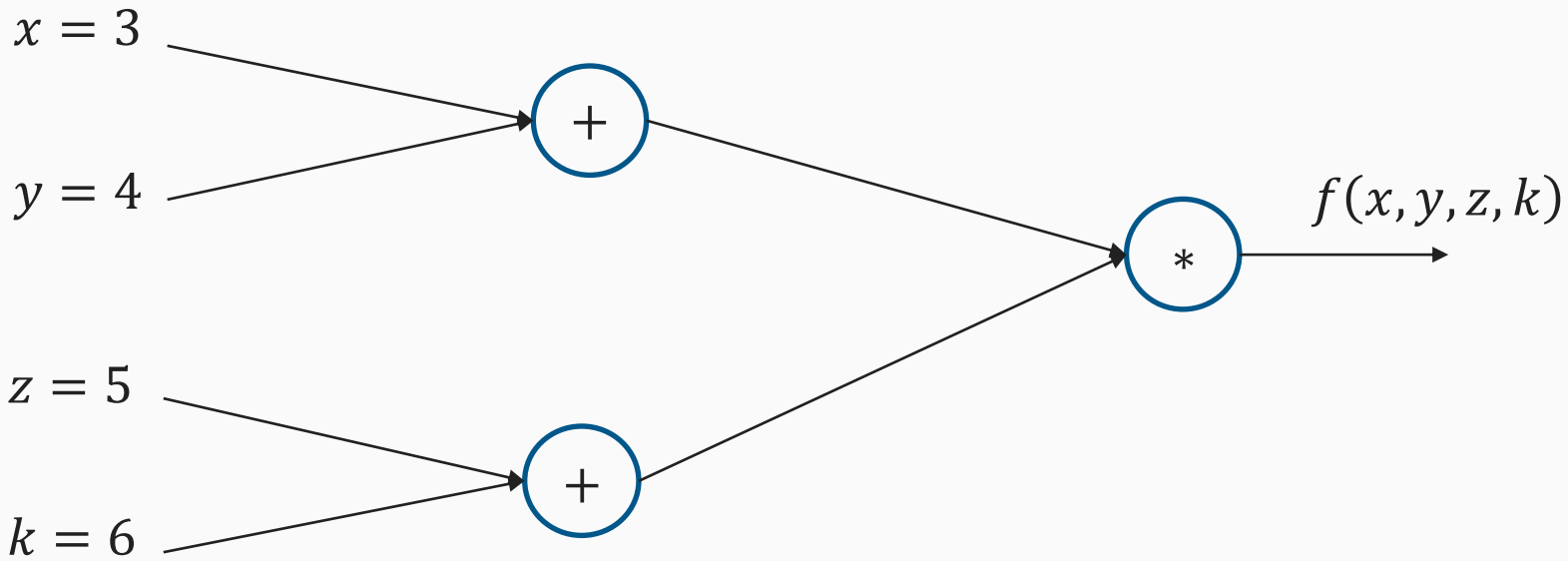
$$f(x, y, z, k) = (x + y)(z + k)$$



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

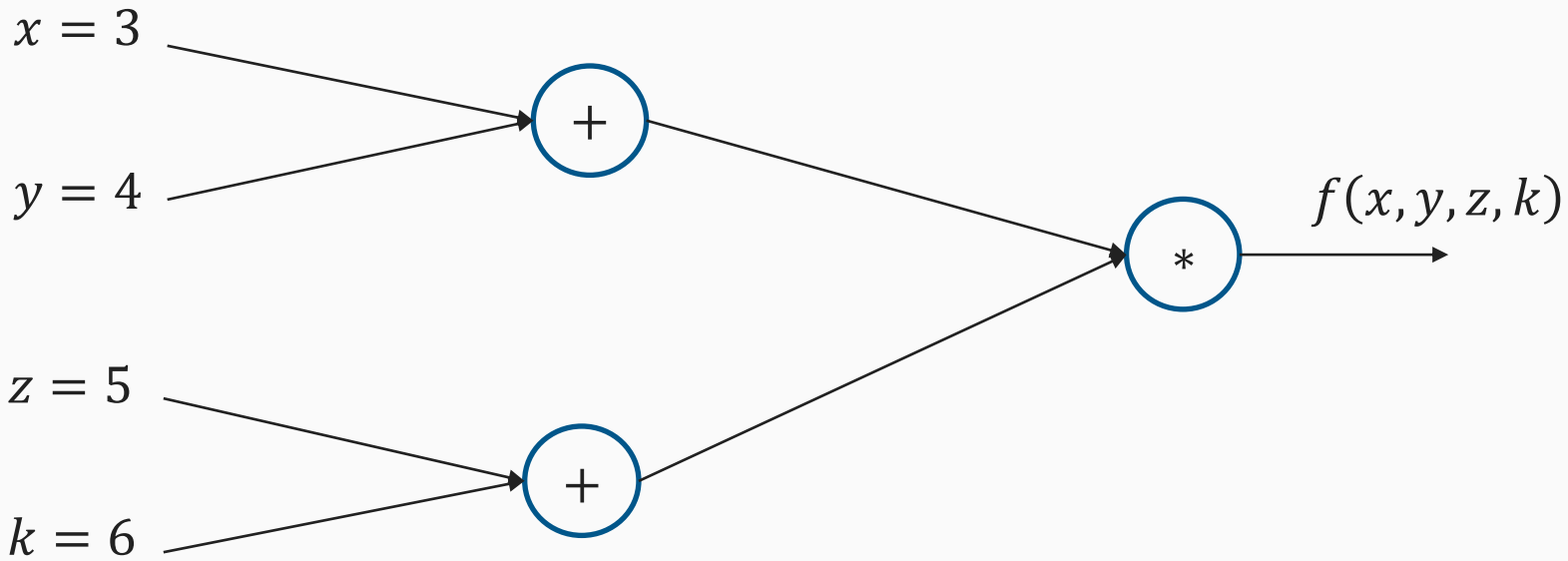


$$f(x, y, z, k) = (x + y)(z + k)$$

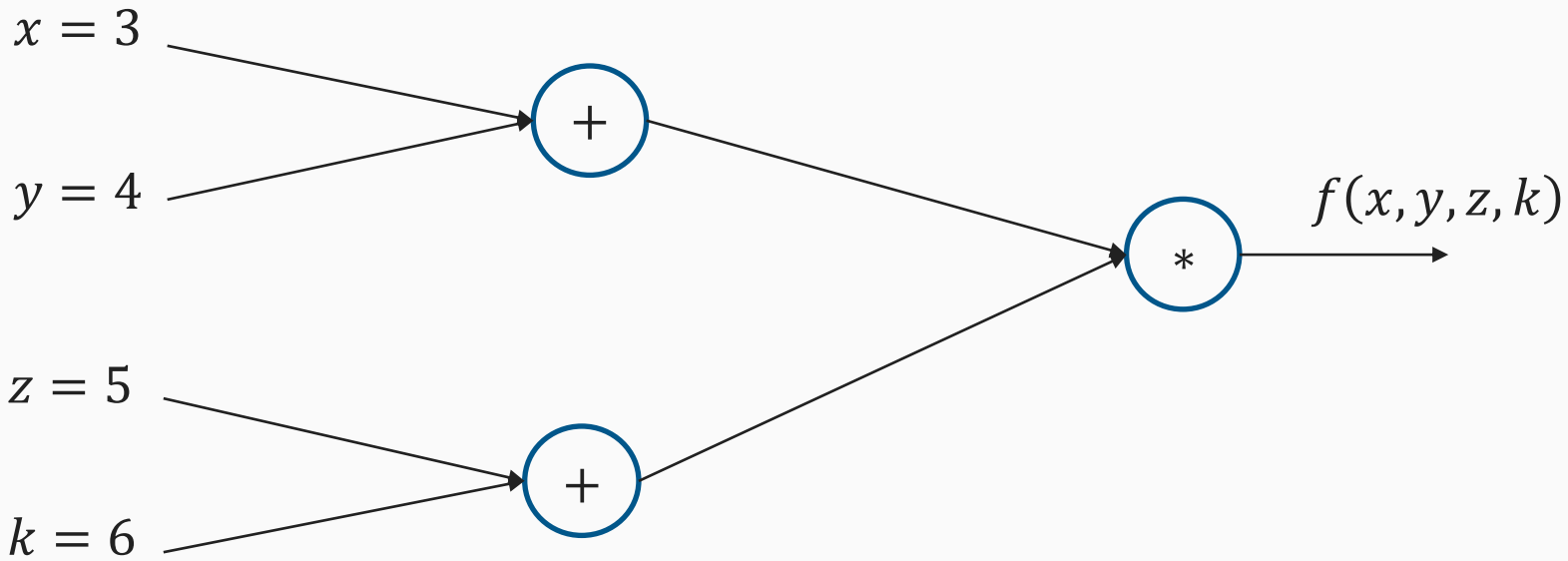


$$f(x, y, z, k) = (x + y)(z + k)$$

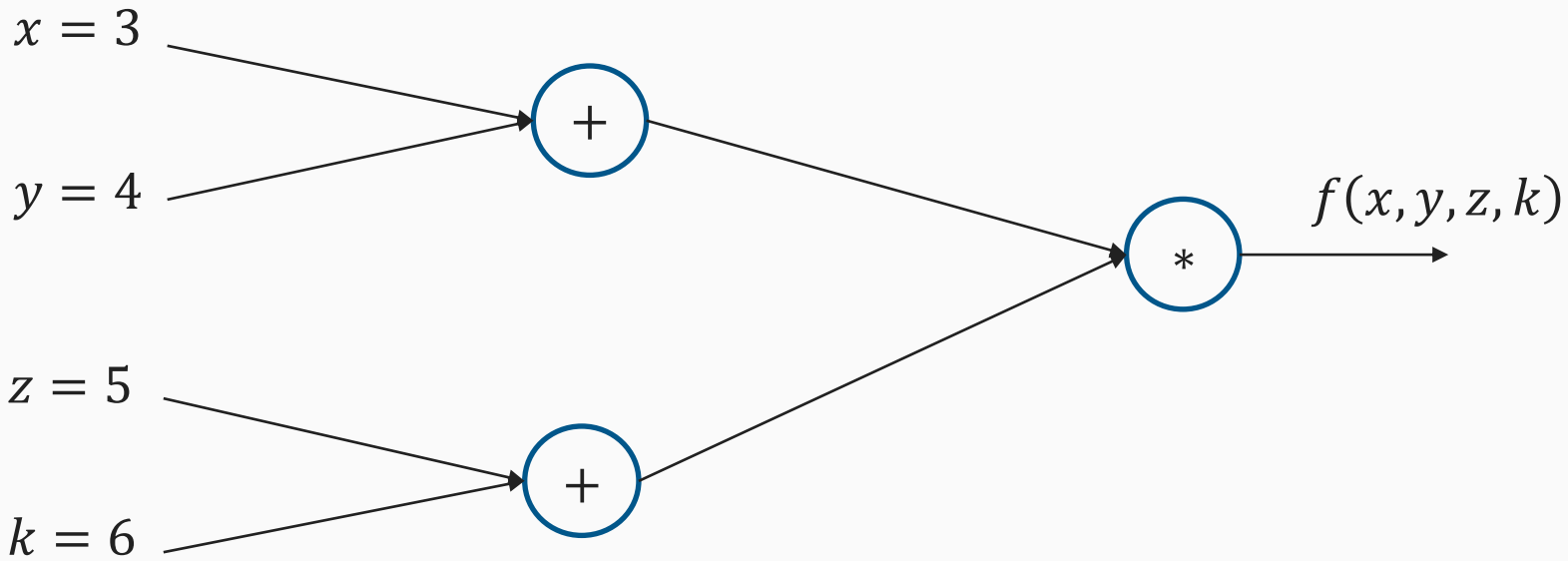
p



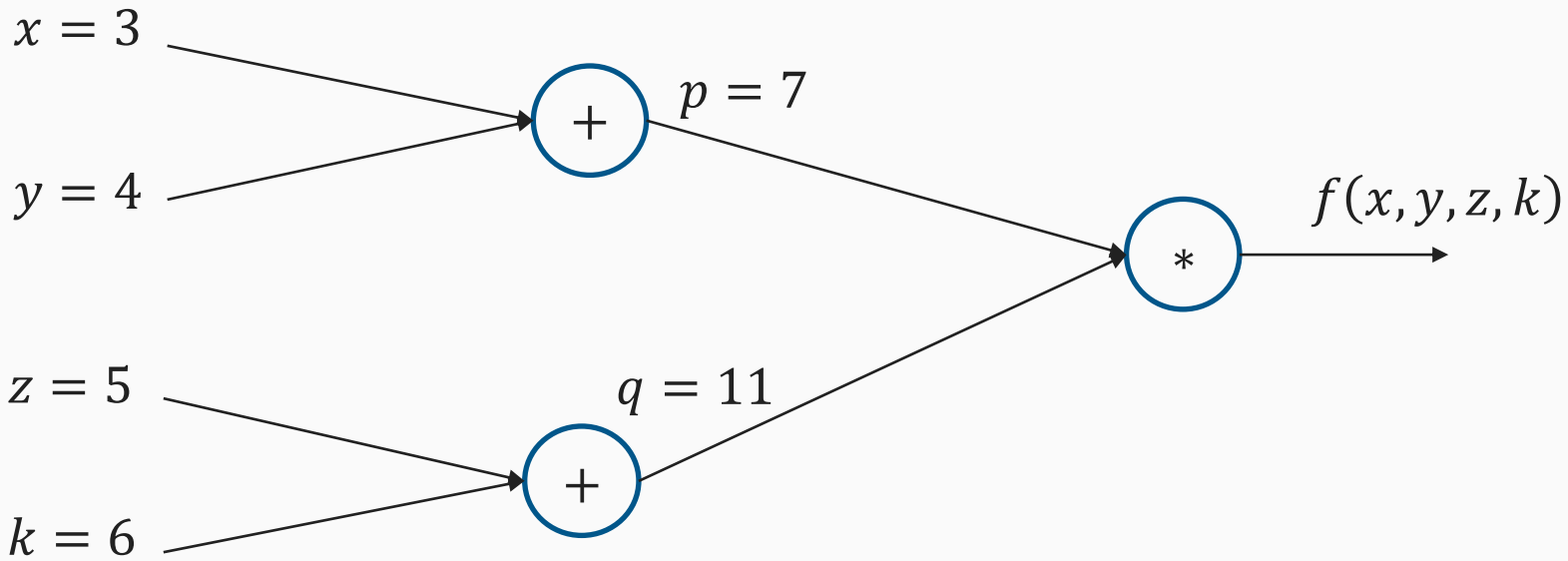
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}(z + k)$$



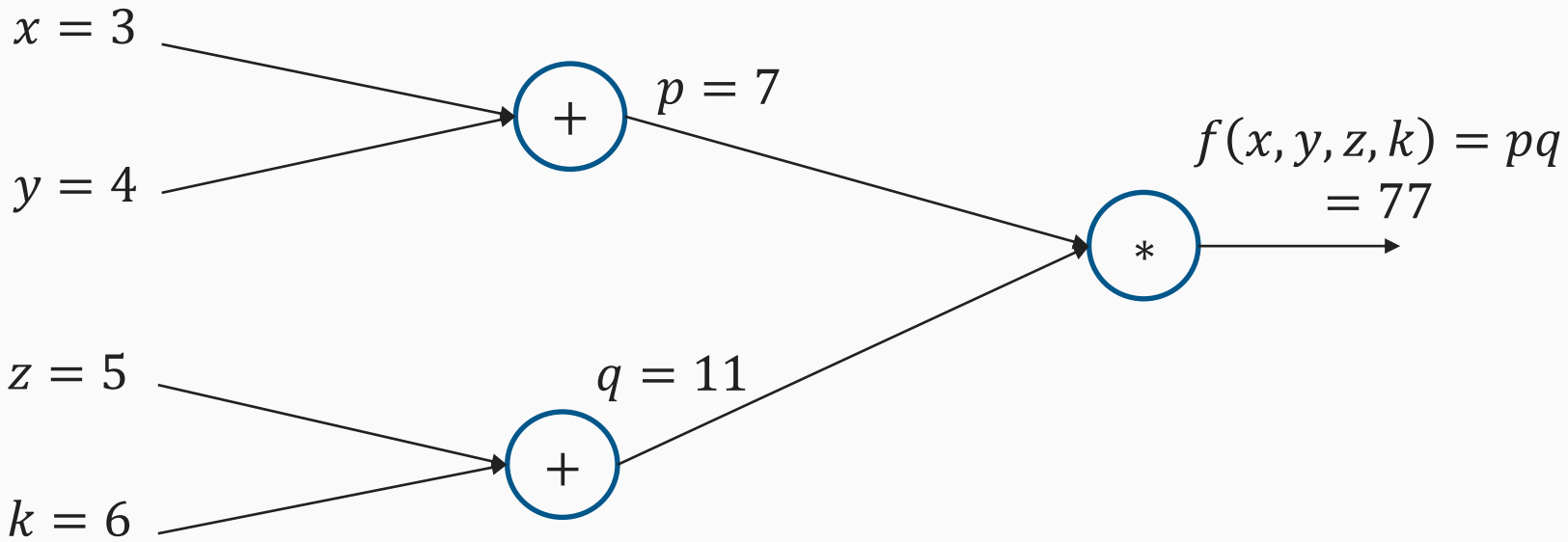
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$



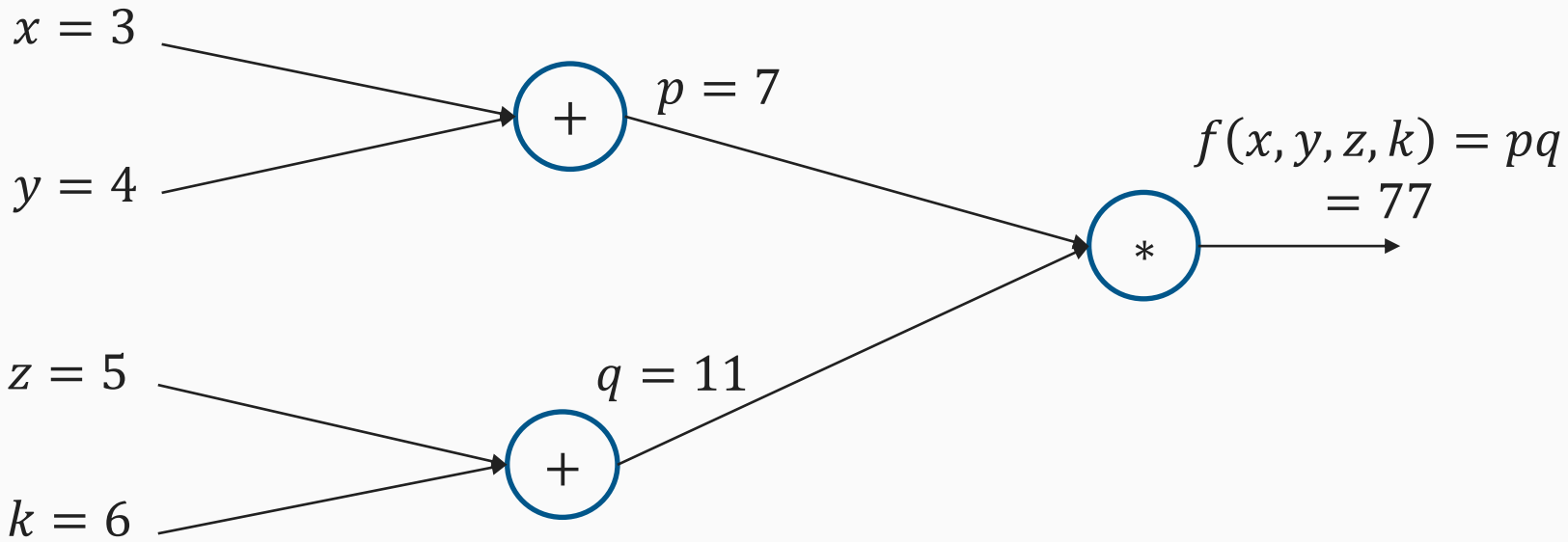
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

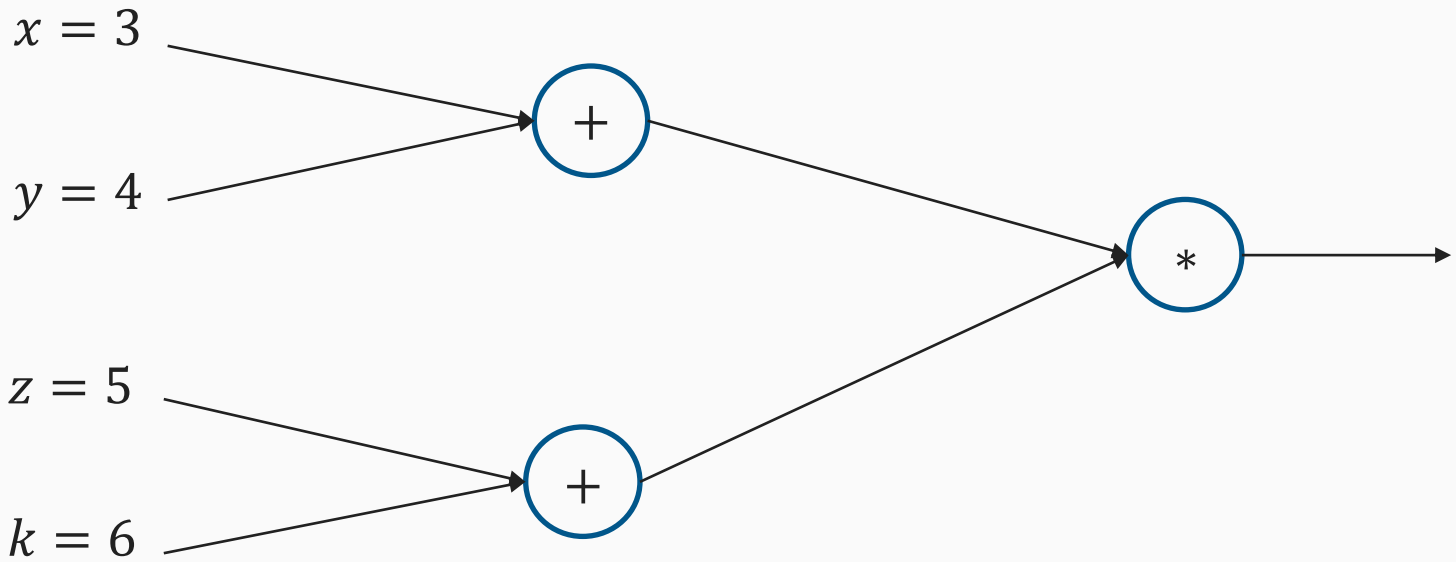


$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$



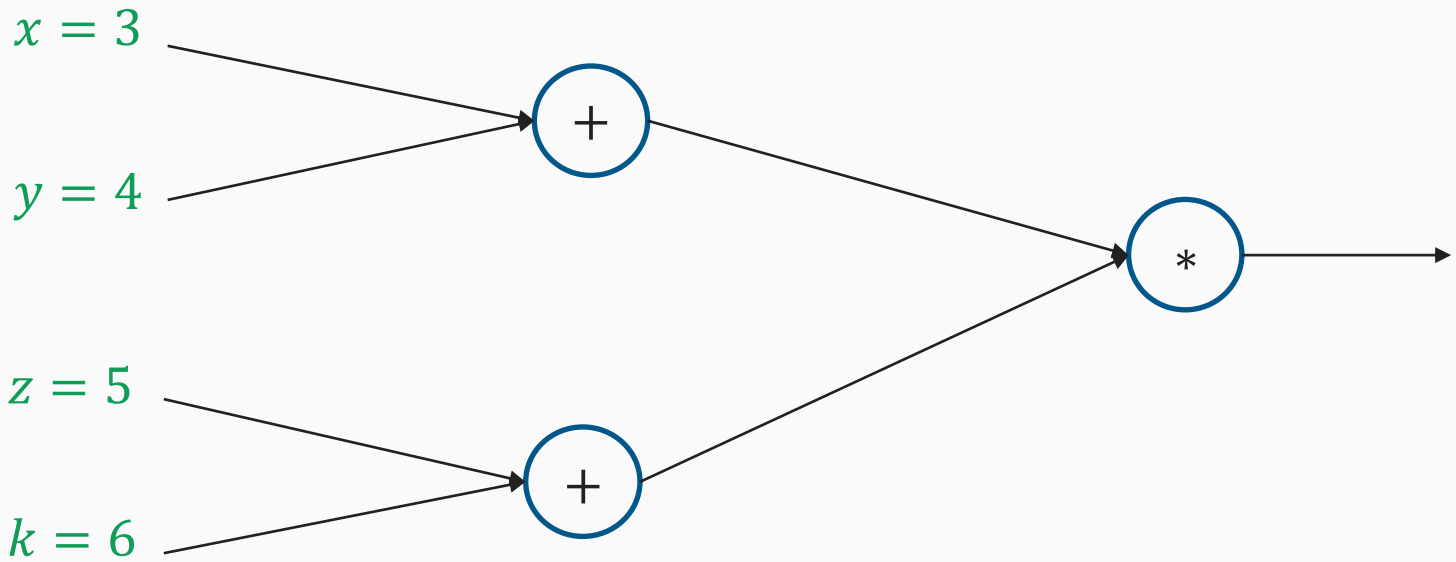
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



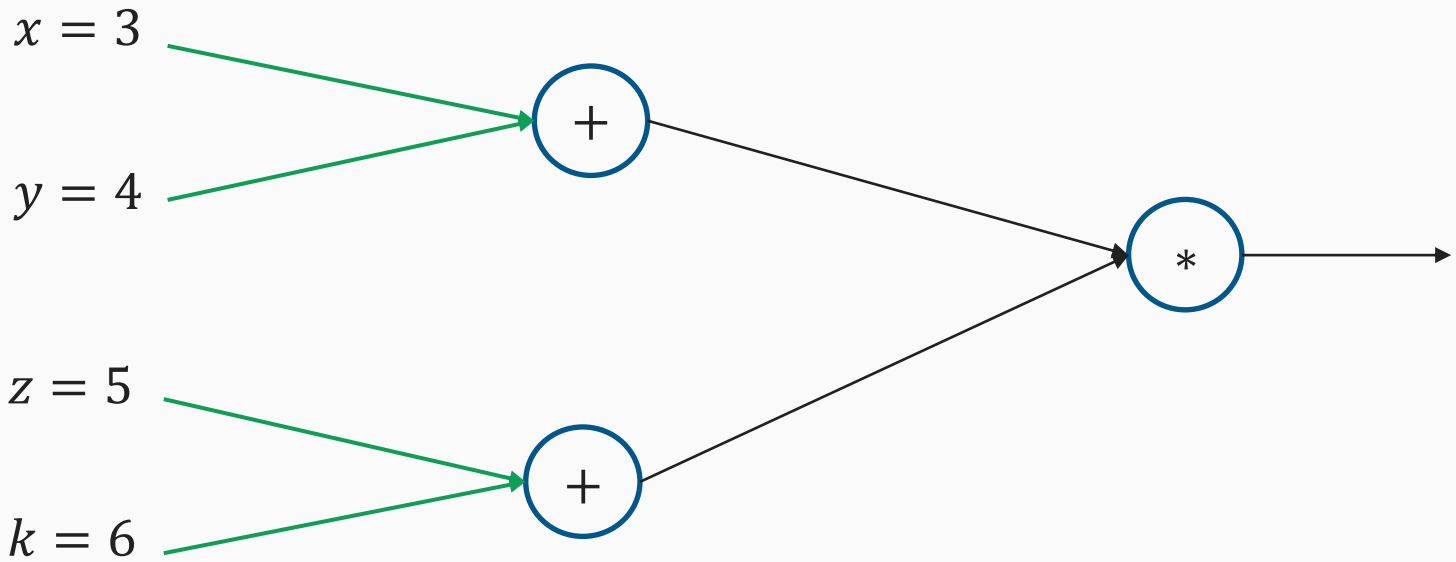
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



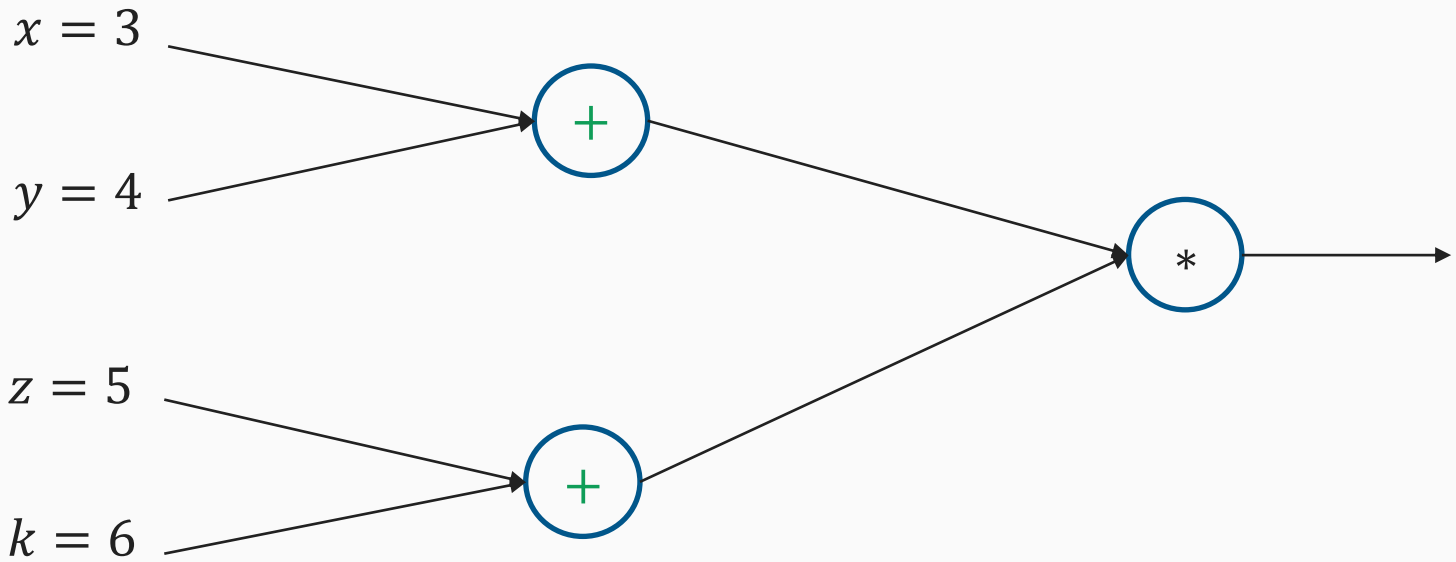
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



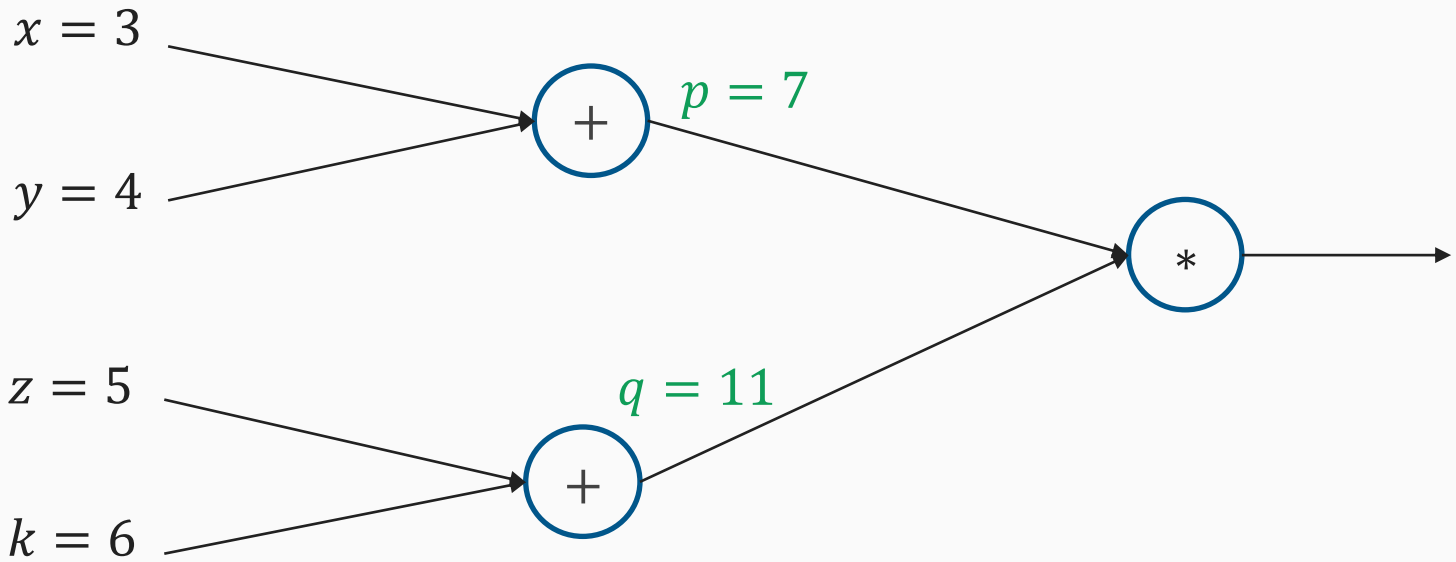
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



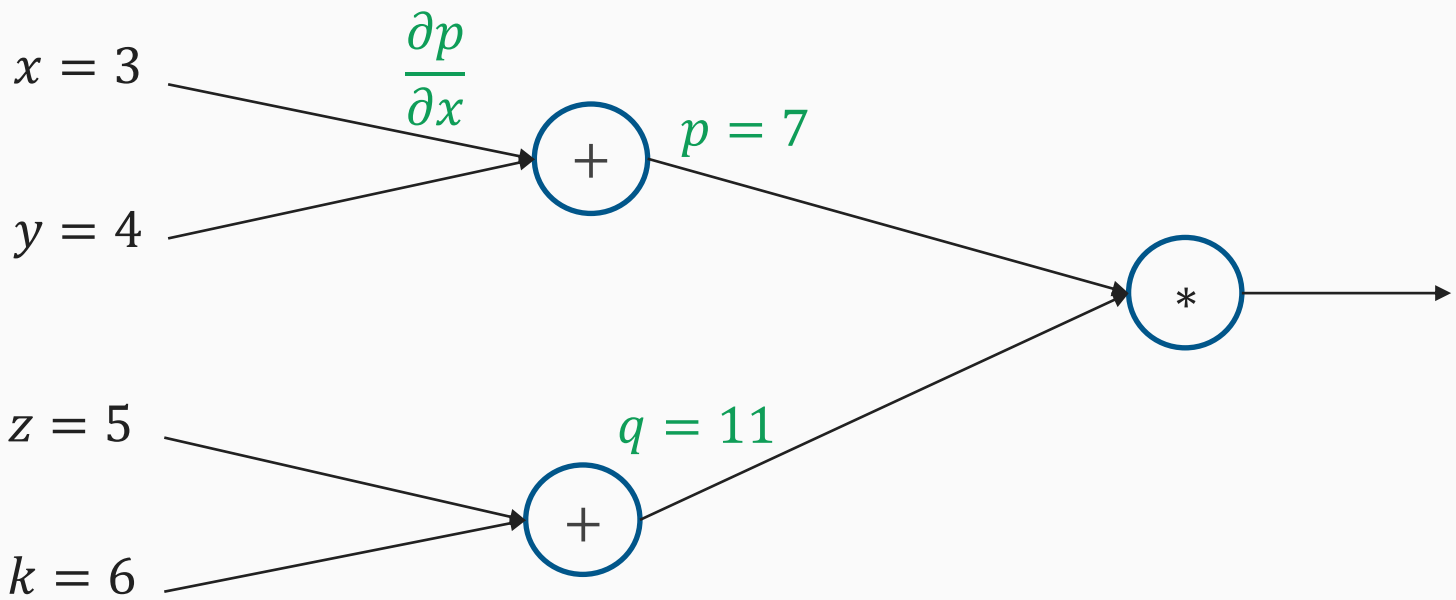
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



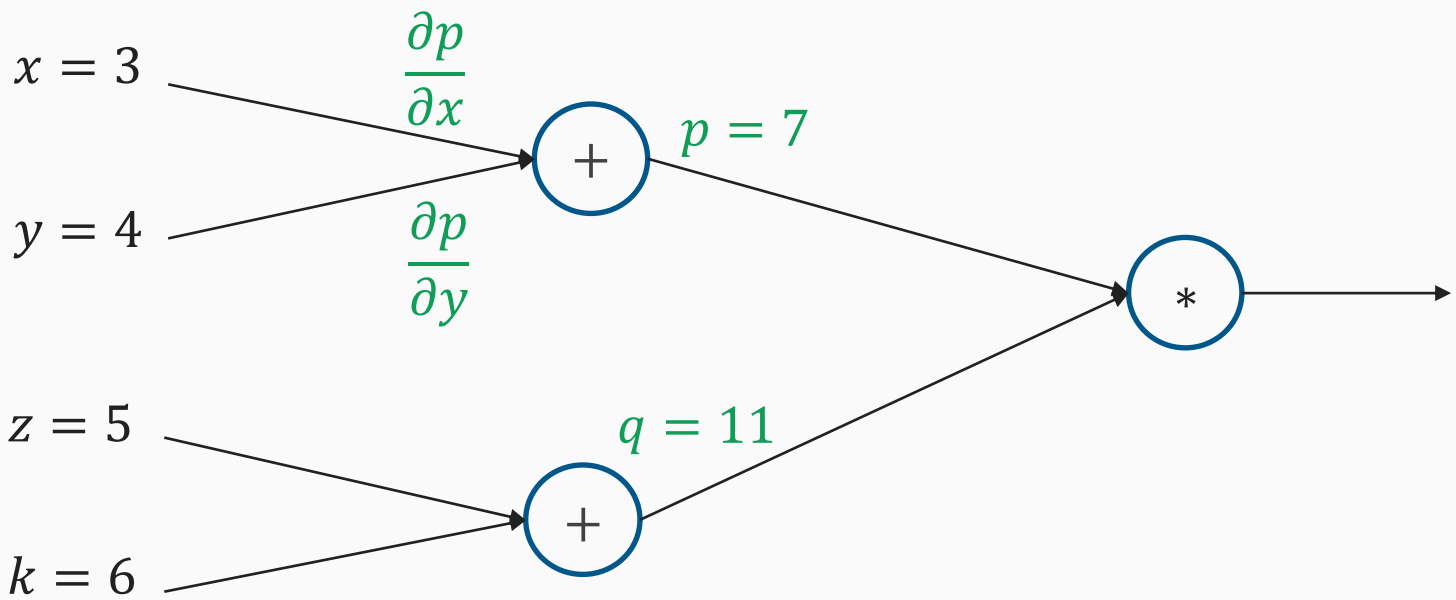
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



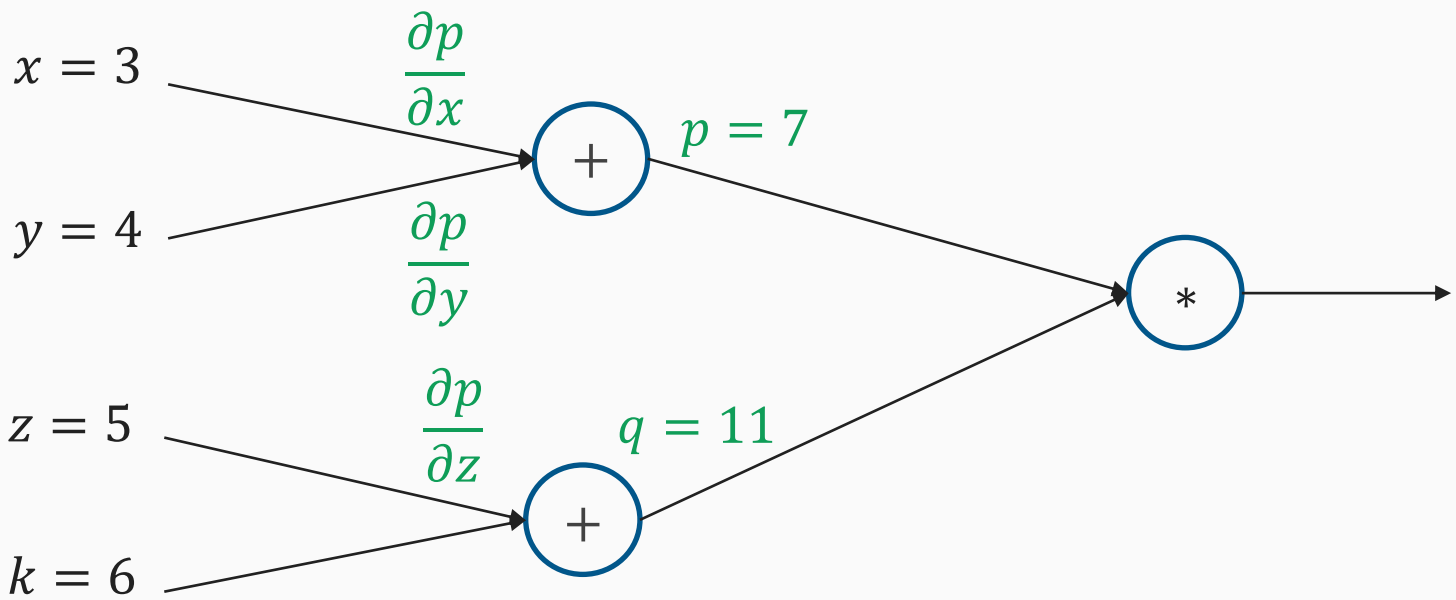
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



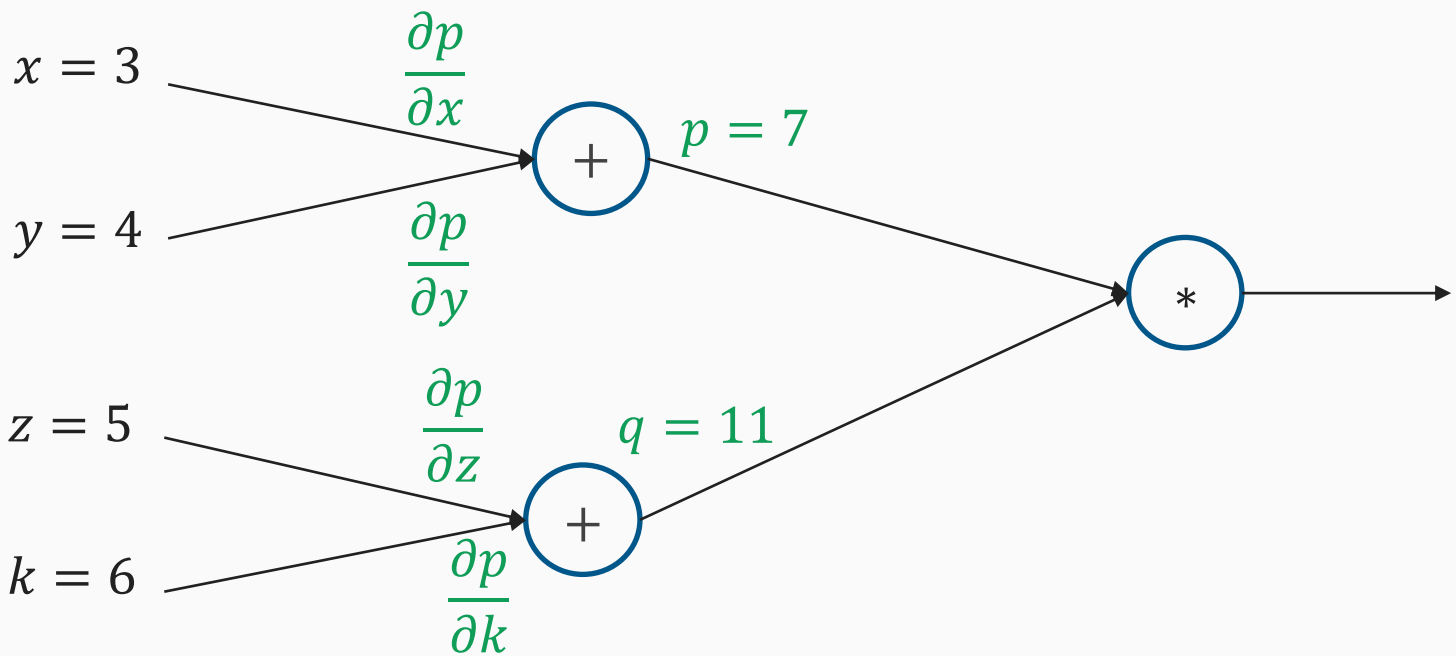
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



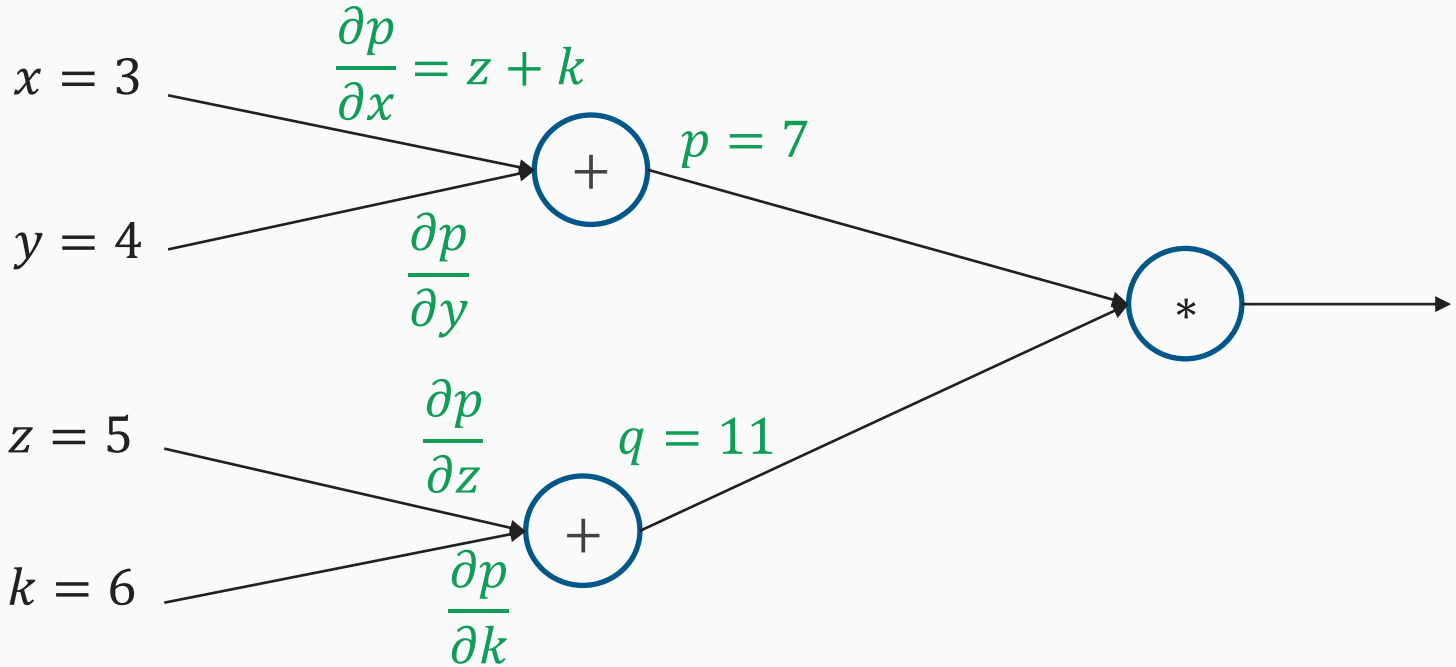
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



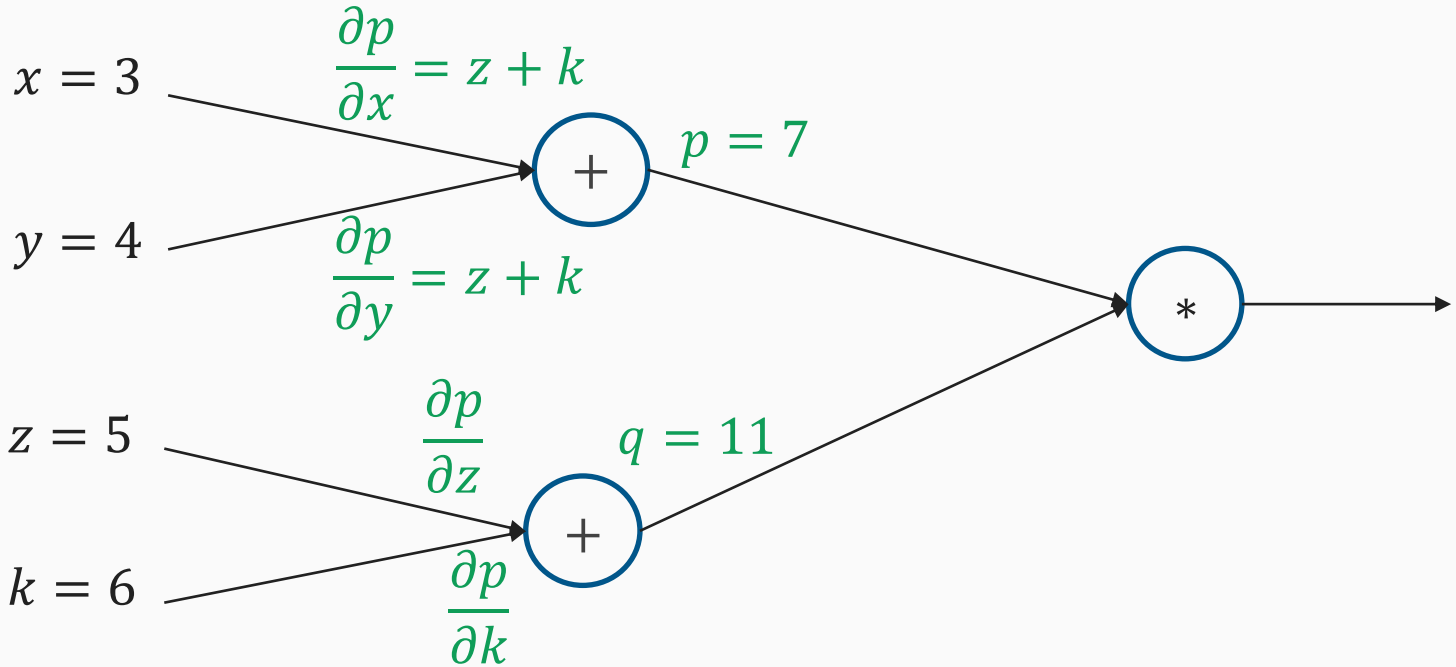
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



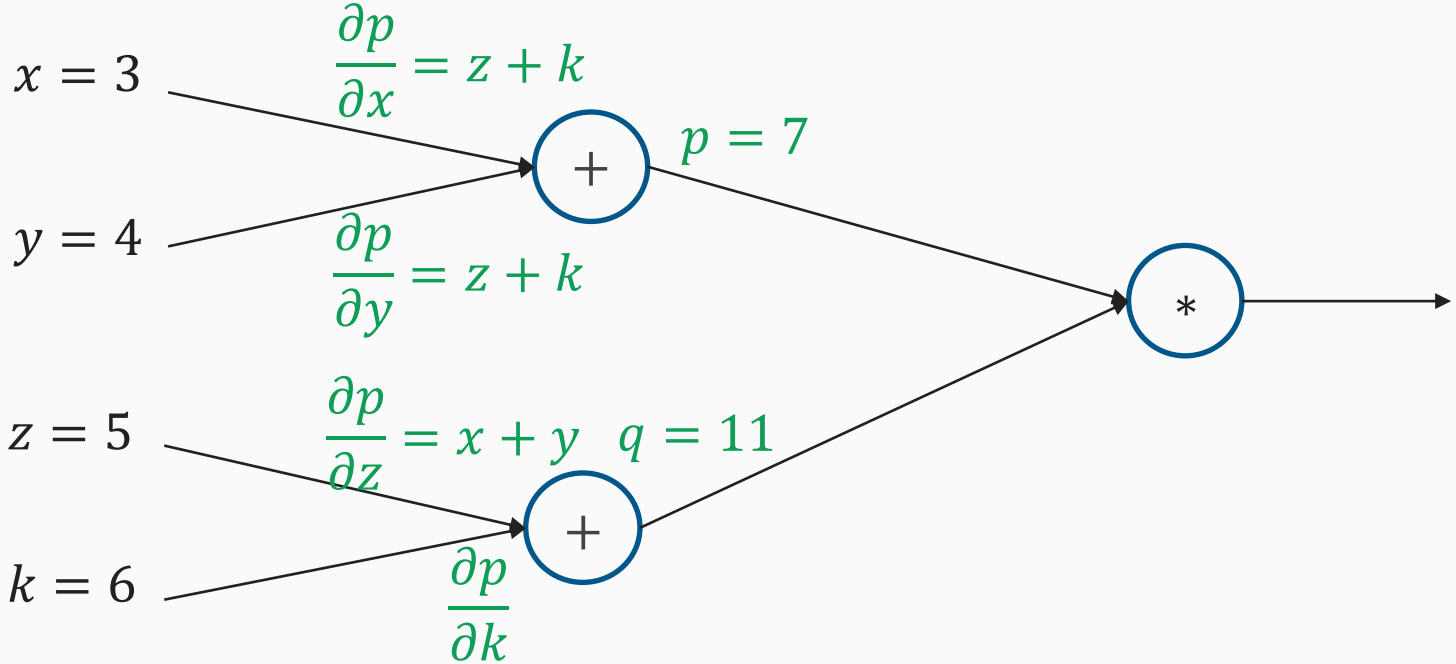
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



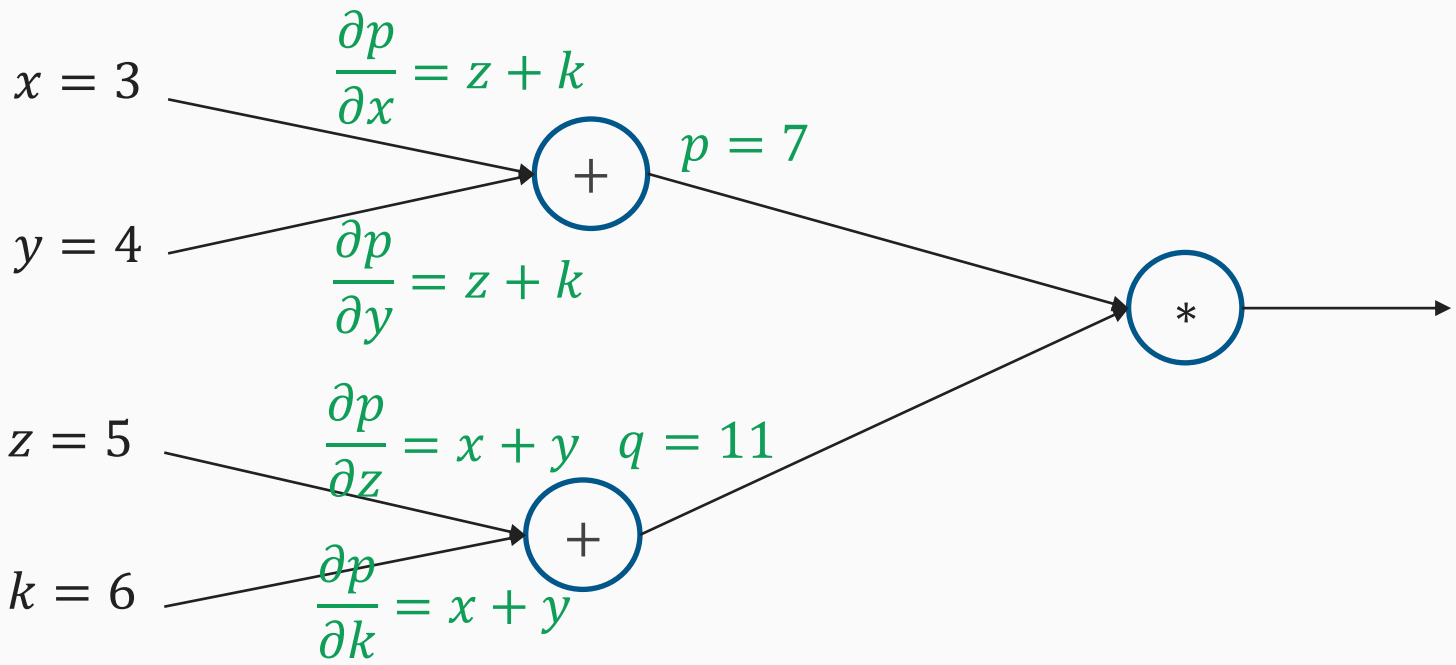
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



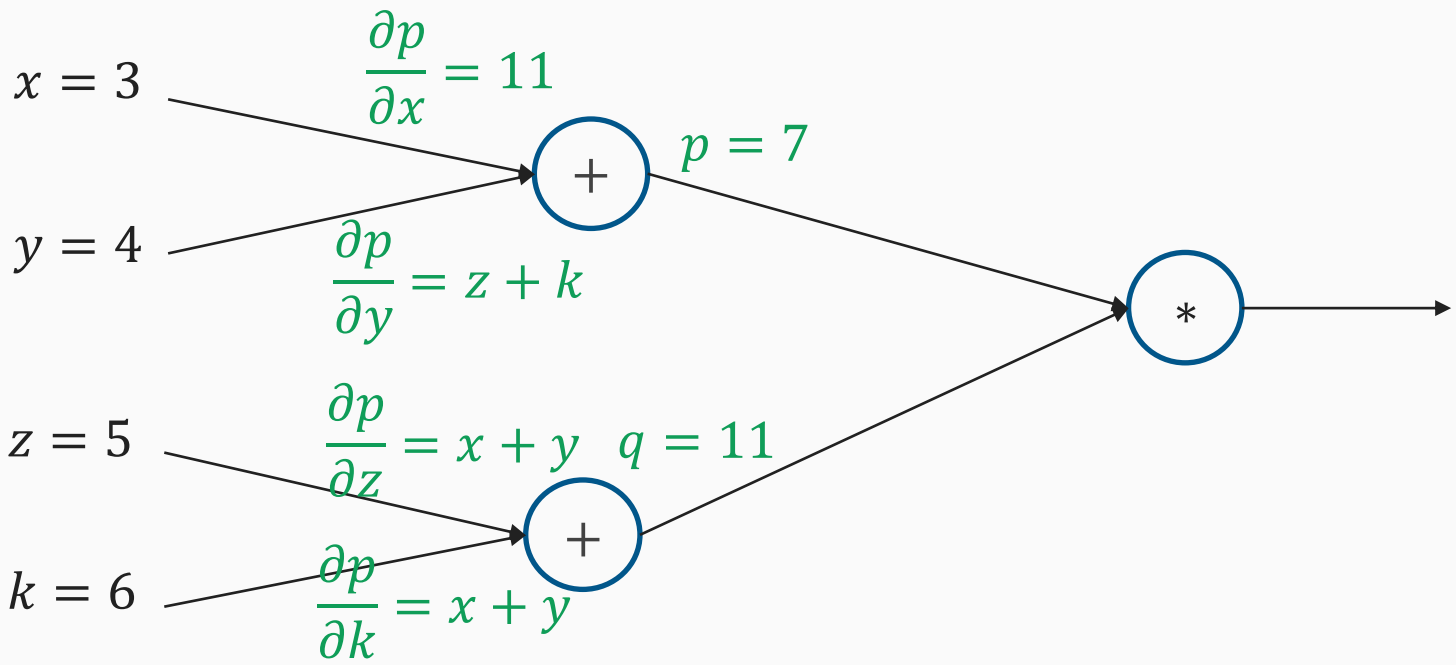
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



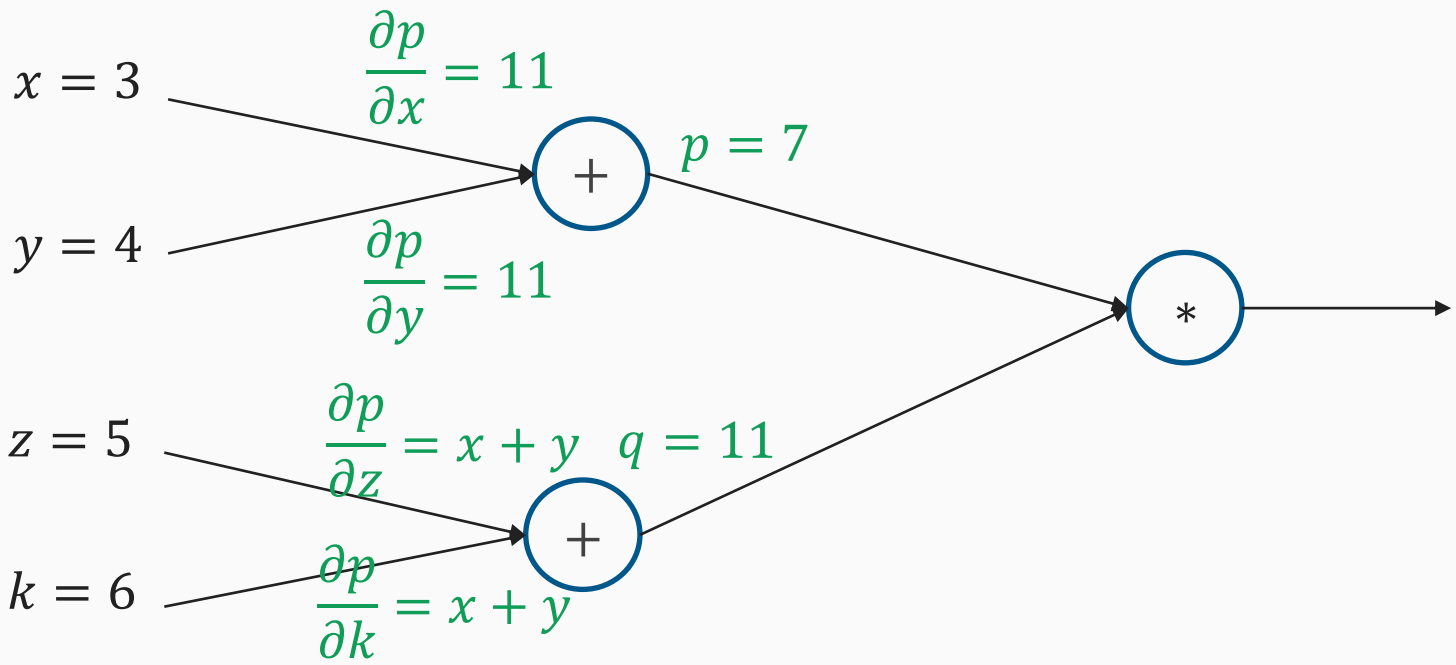
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



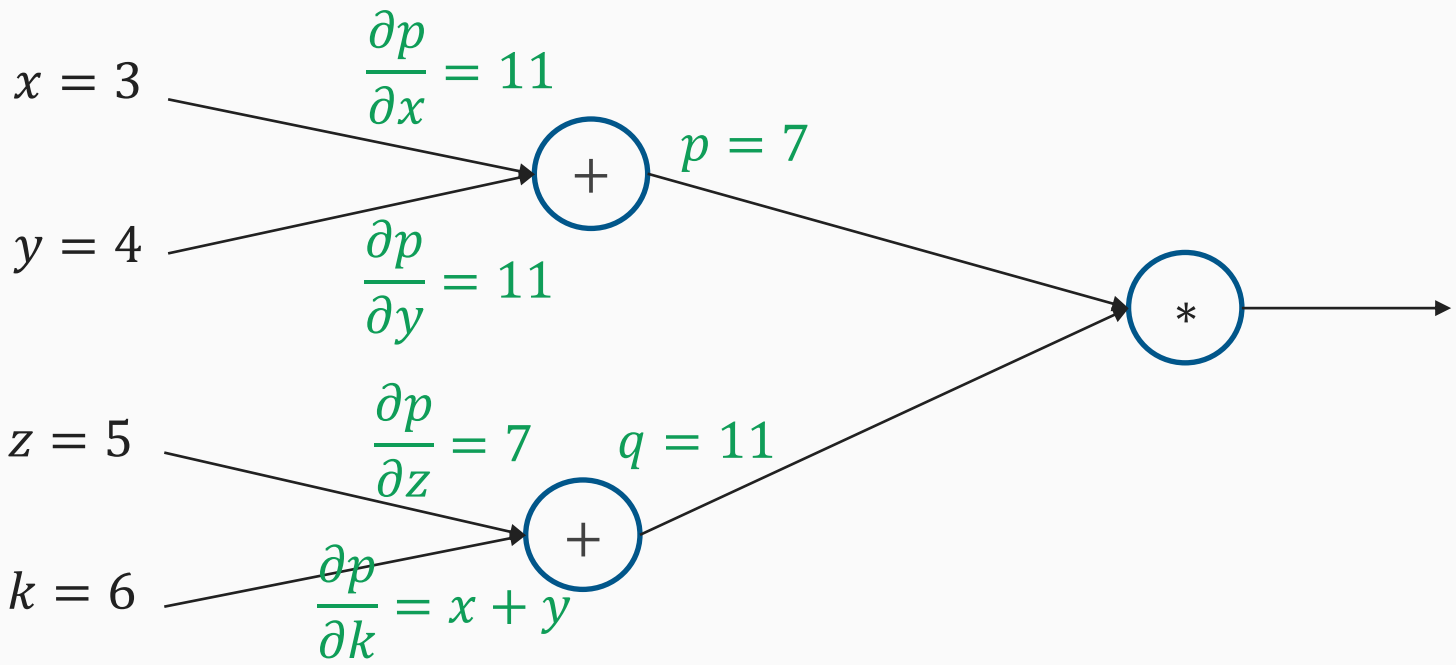
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



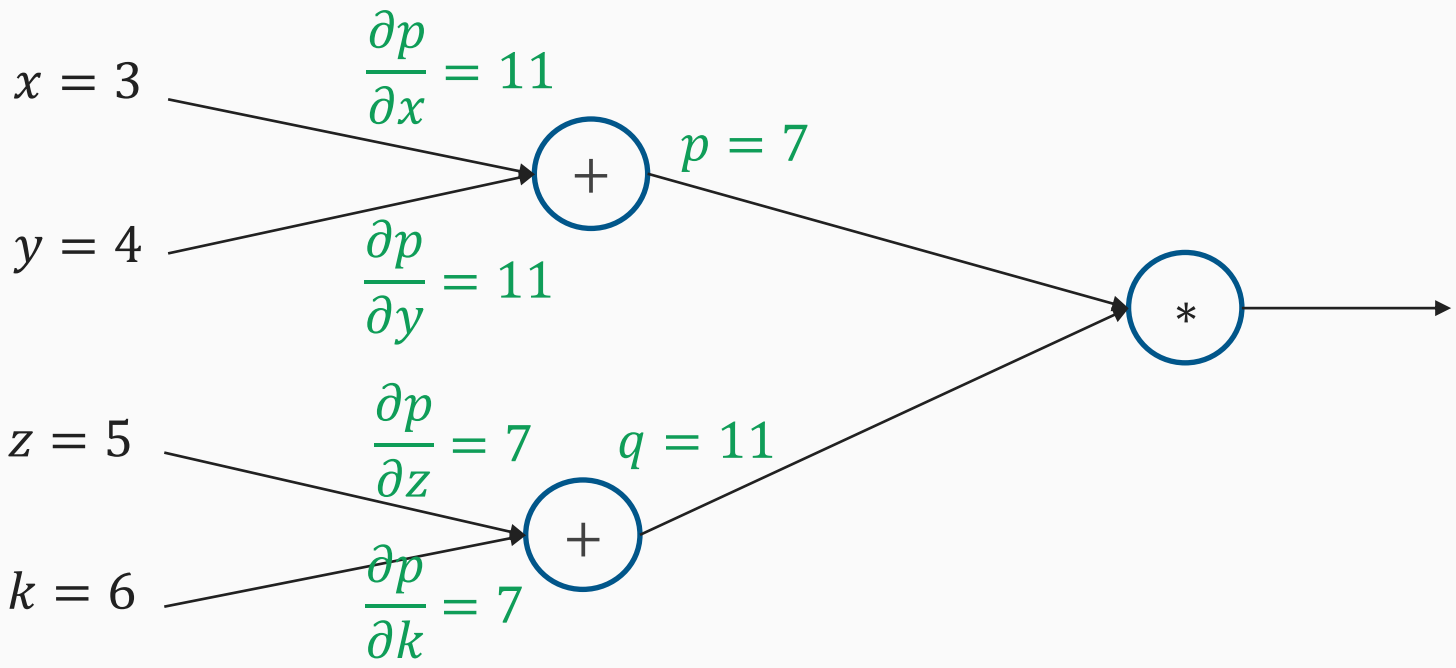
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

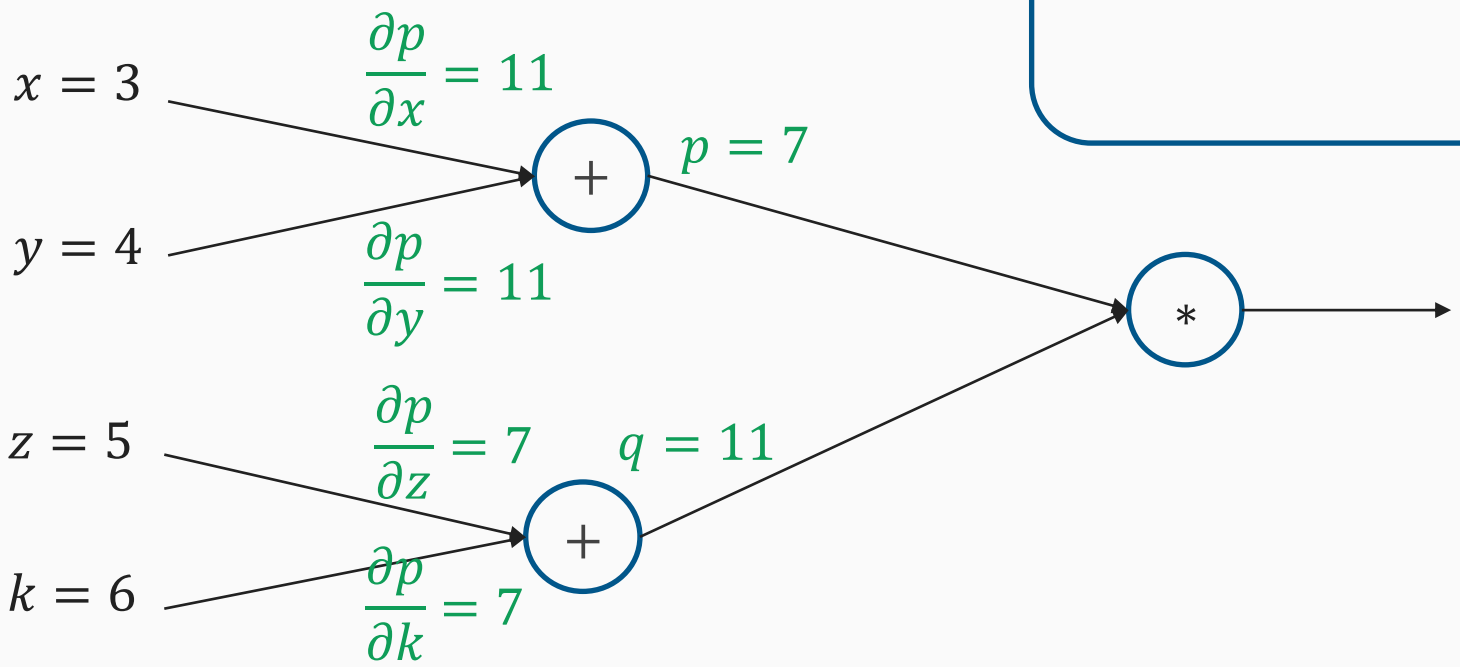
Forward pass:



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory



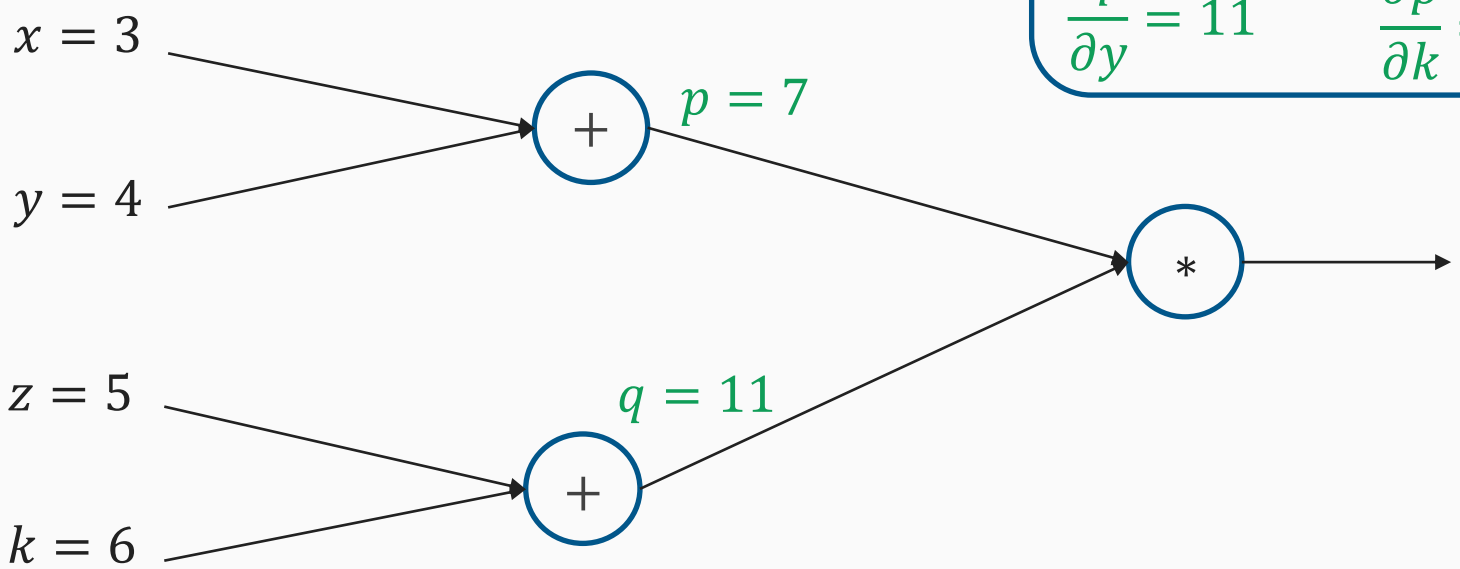
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$



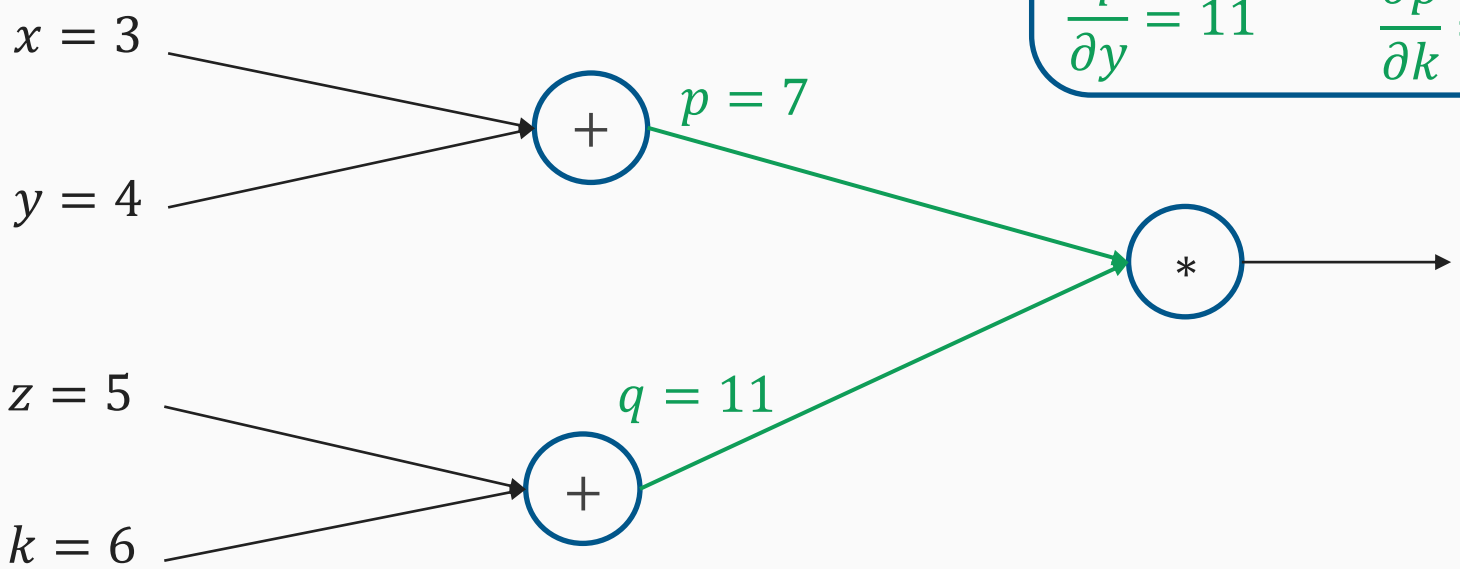
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$



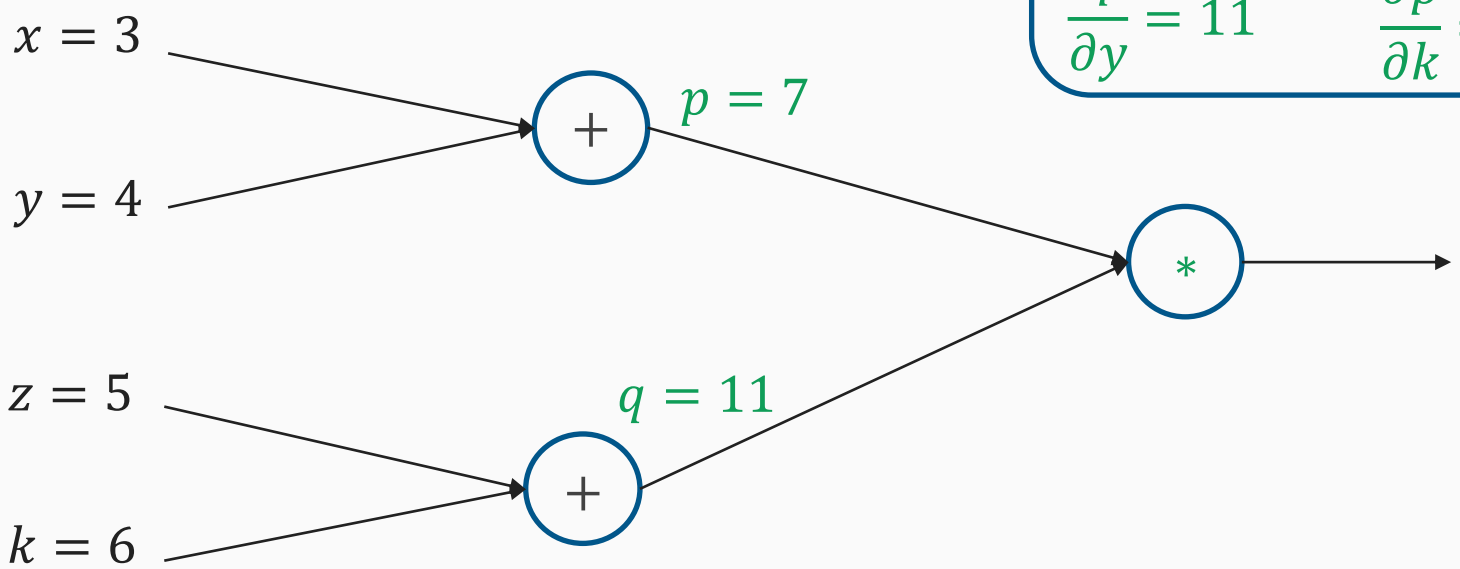
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$



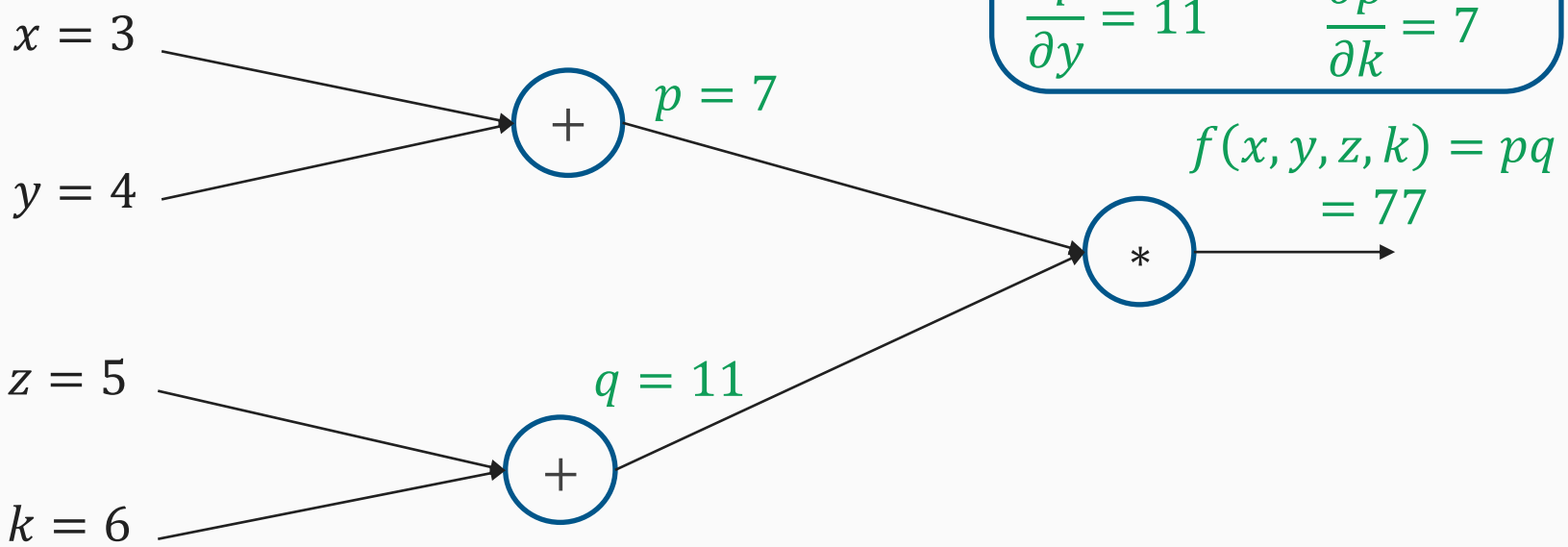
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$



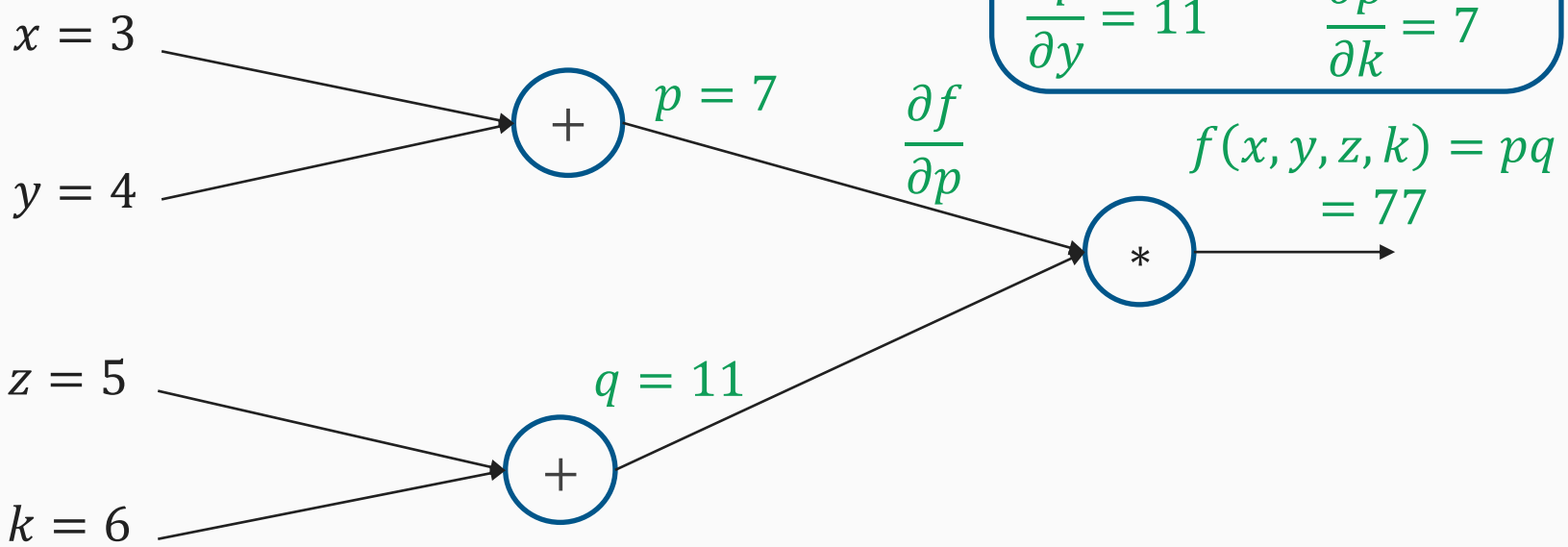
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$



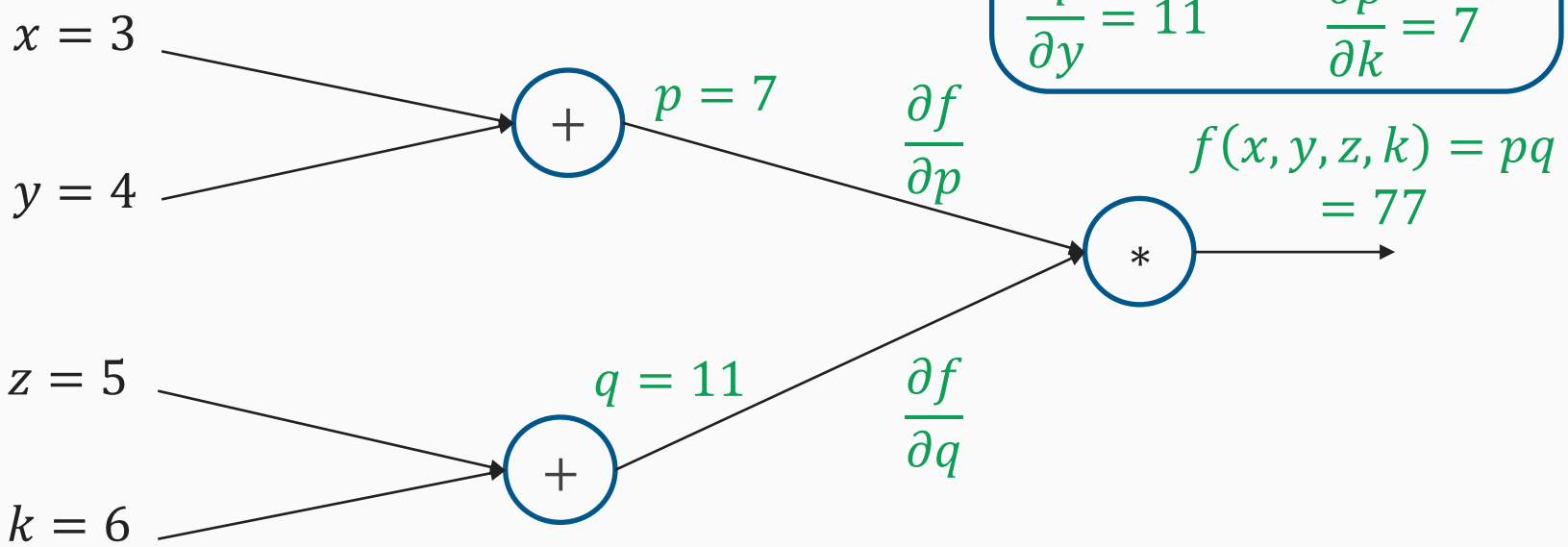
$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

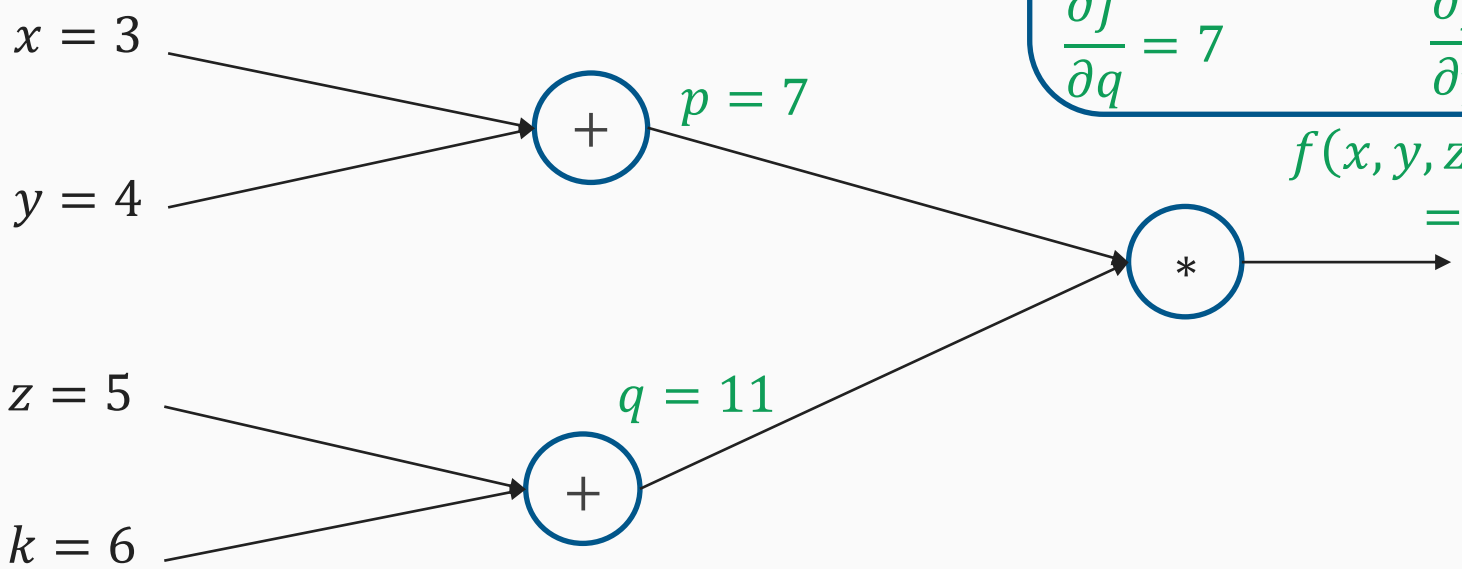
Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = pq$$
$$= 77$$

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

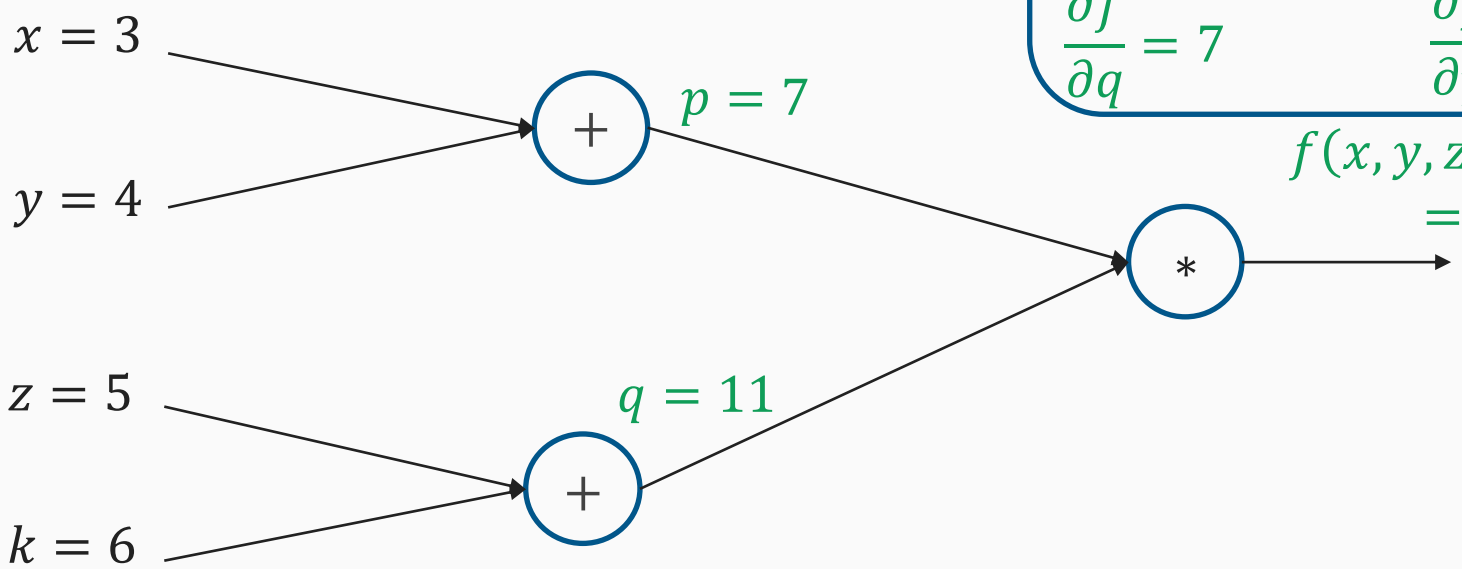
Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = pq$$
$$= 77$$

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

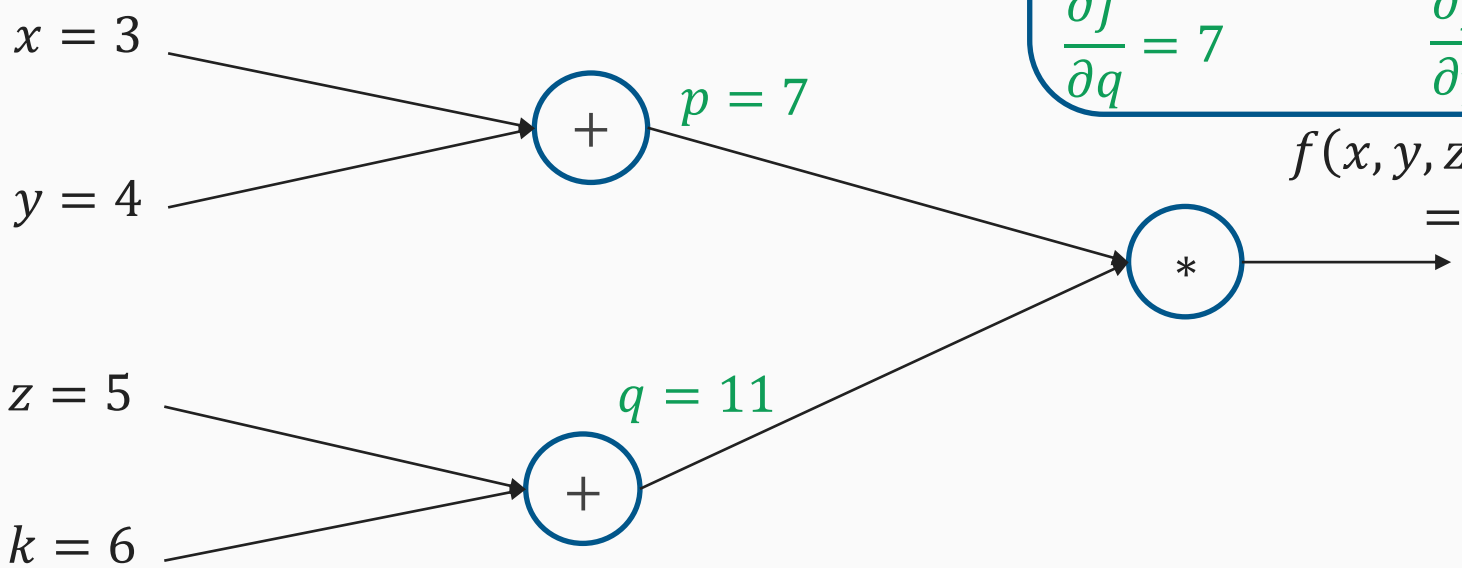
Forward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

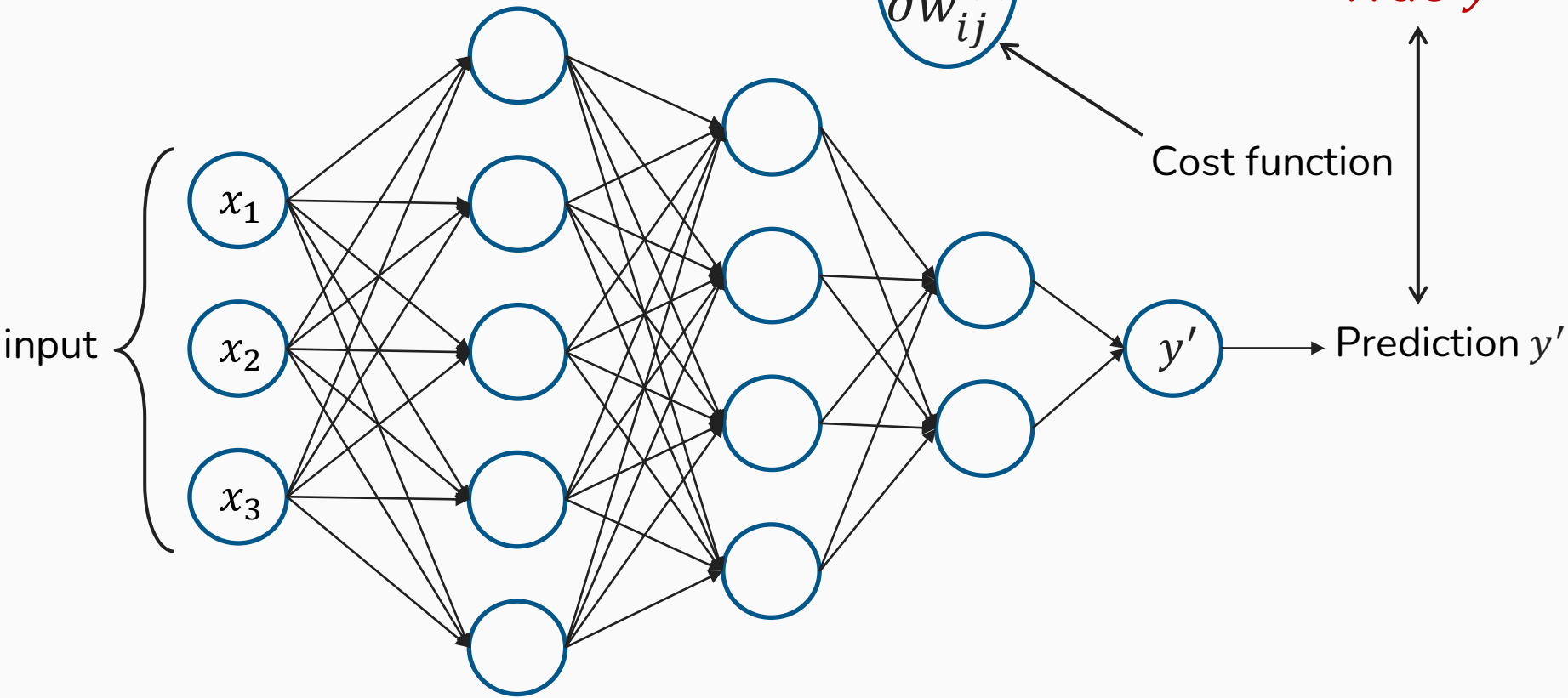
$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$

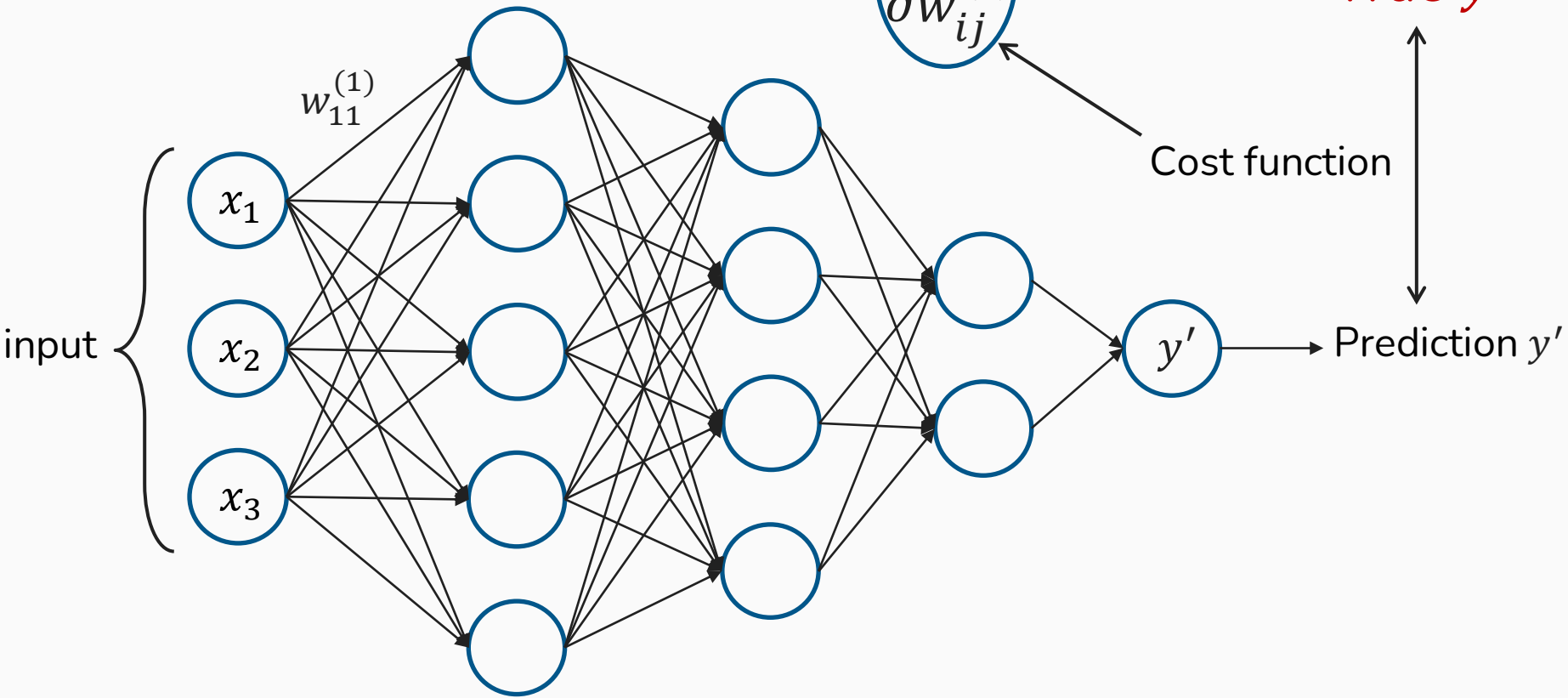


$$f(x, y, z, k) = pq$$
$$= 77$$

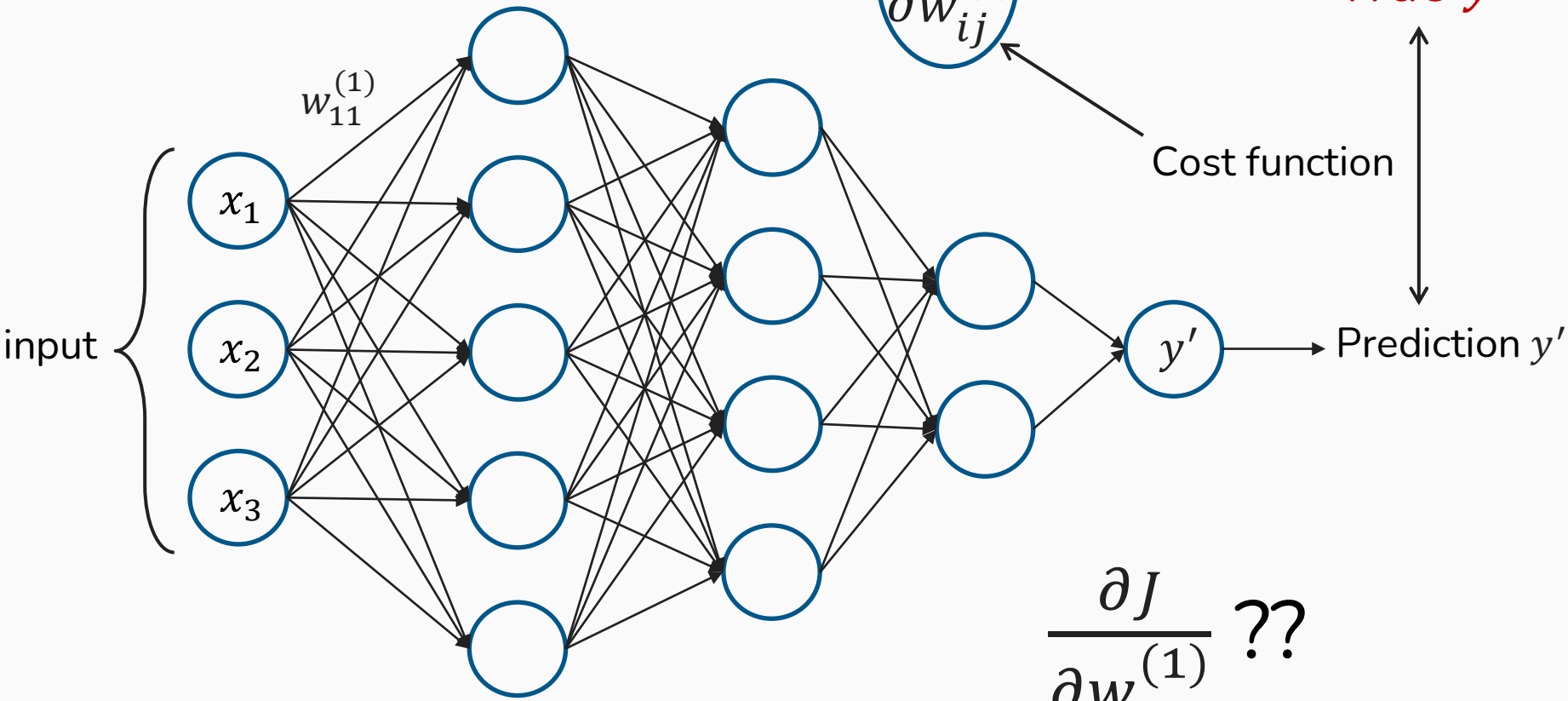
$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$



$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$



$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$



$$\frac{\partial J}{\partial w_{11}^{(1)}} \; ??$$

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

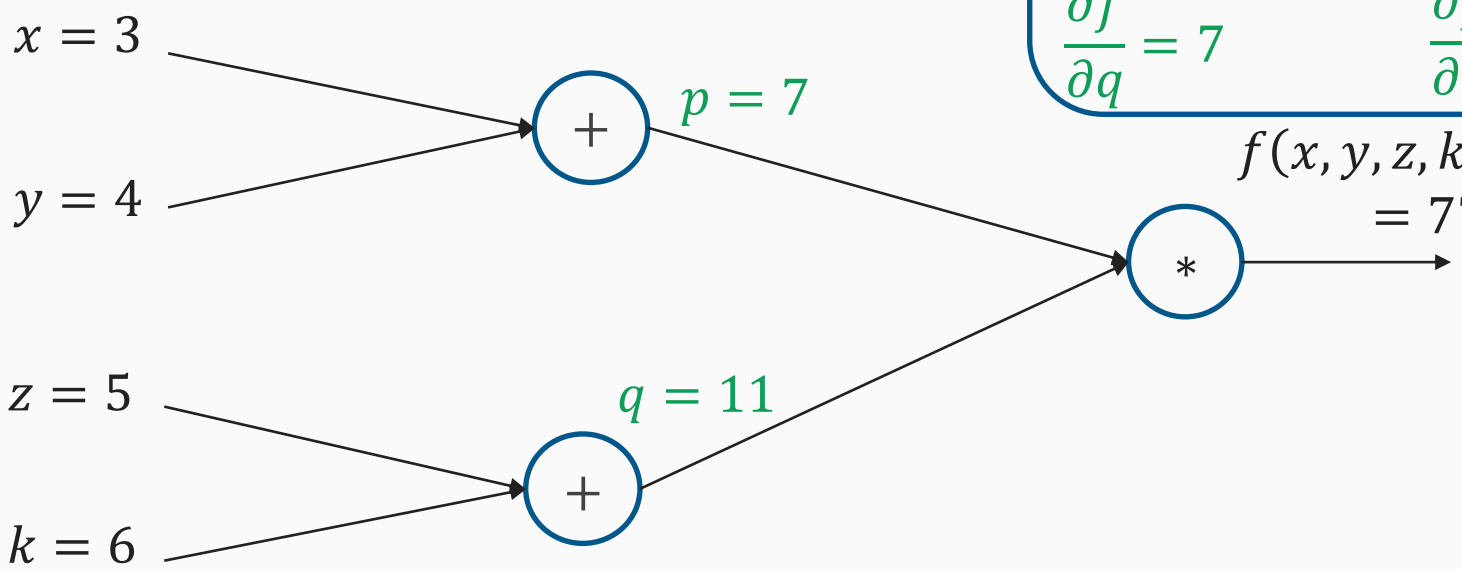
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = pq$$
$$= 77$$

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

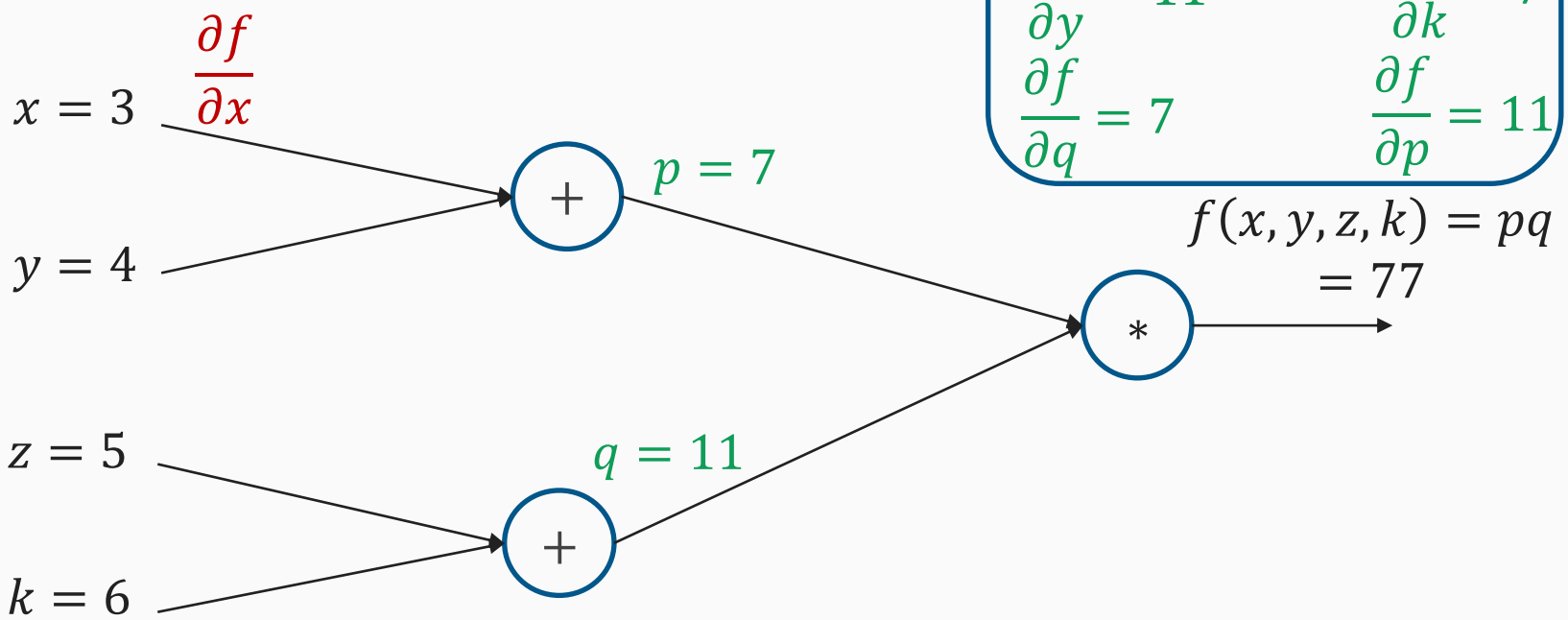
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

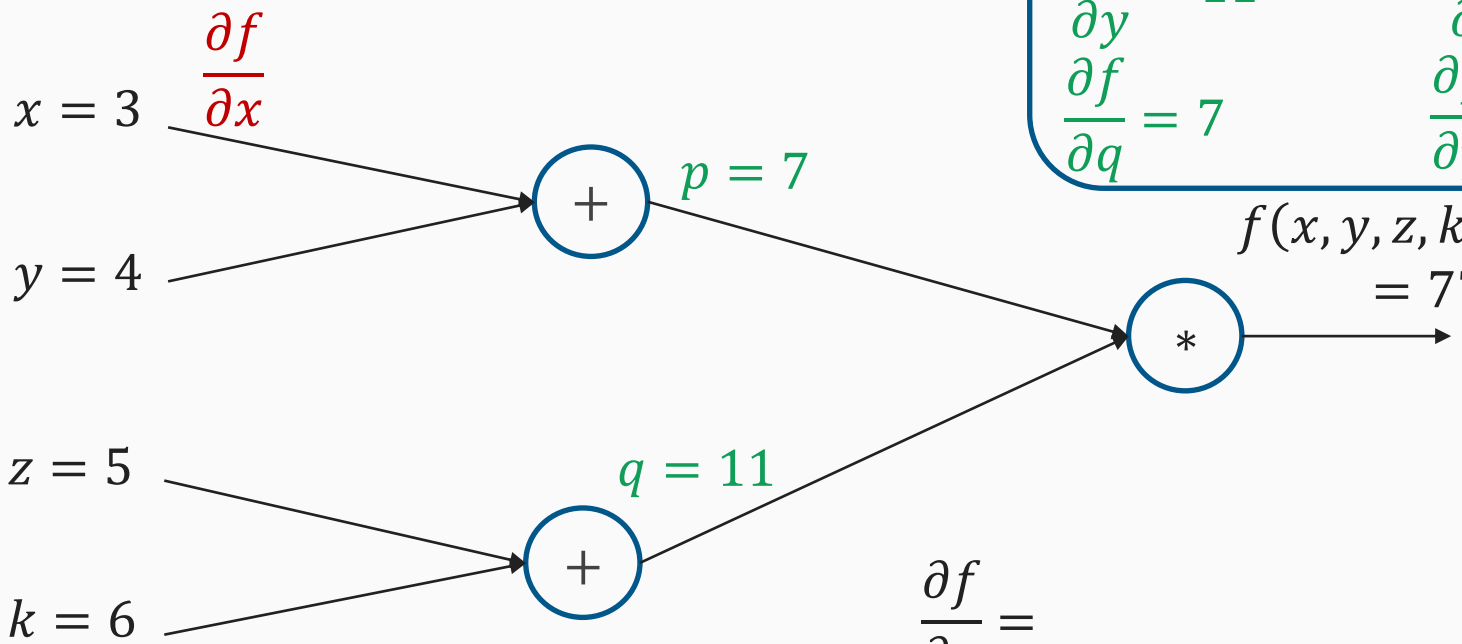
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = pq$$
$$= 77$$

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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

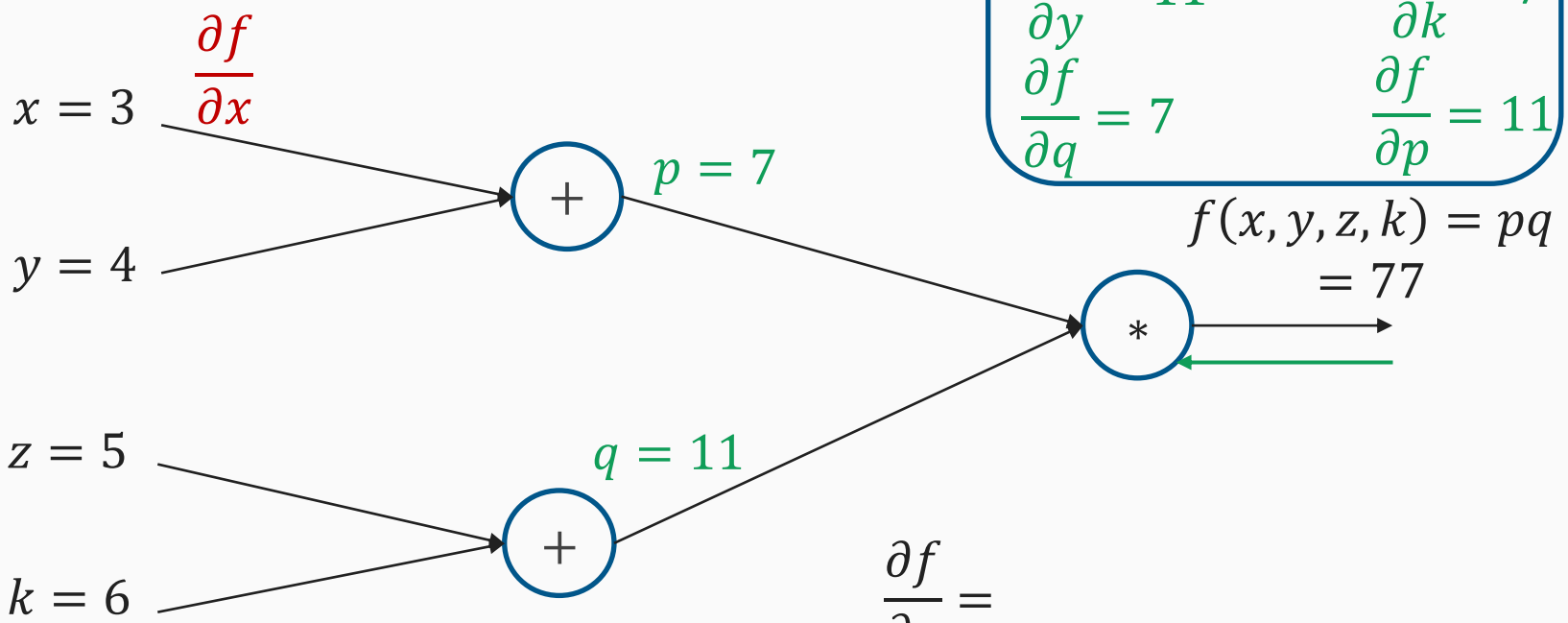
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

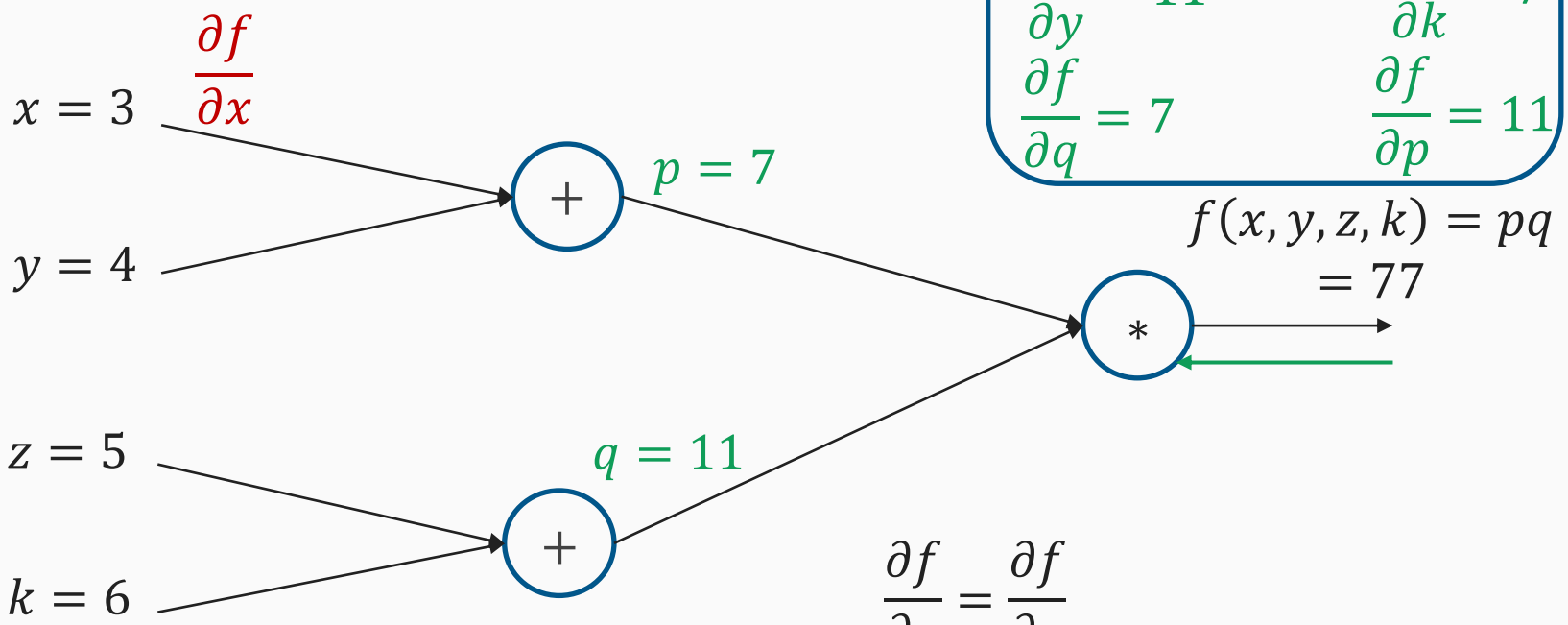
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

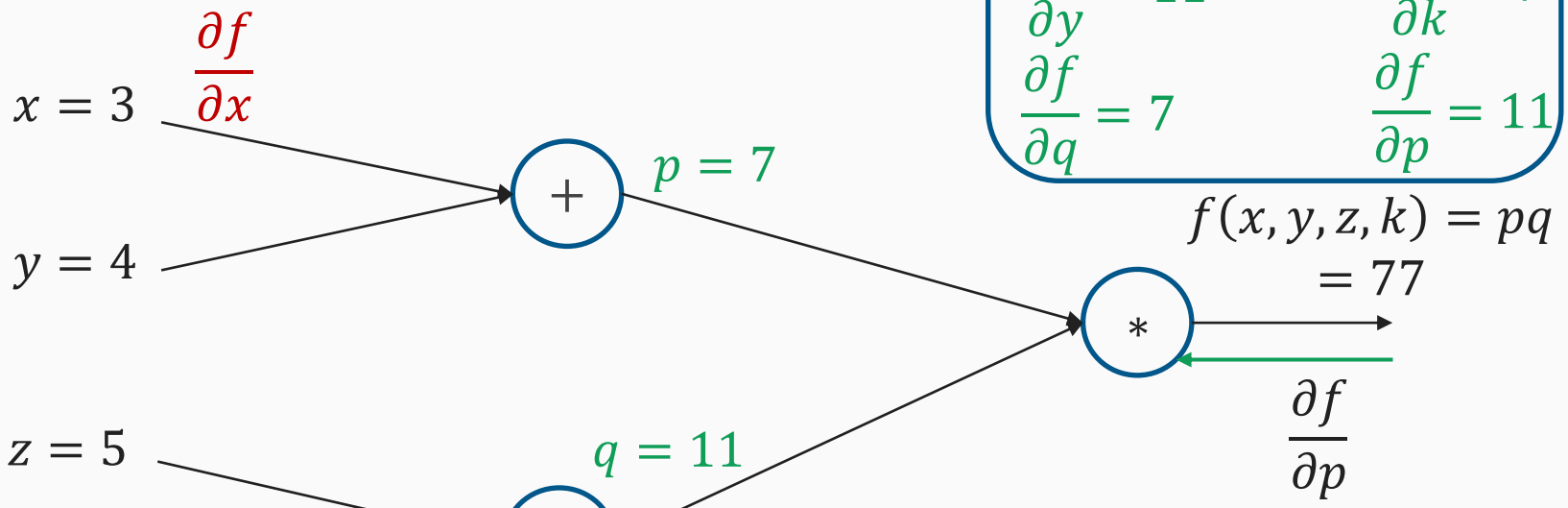
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

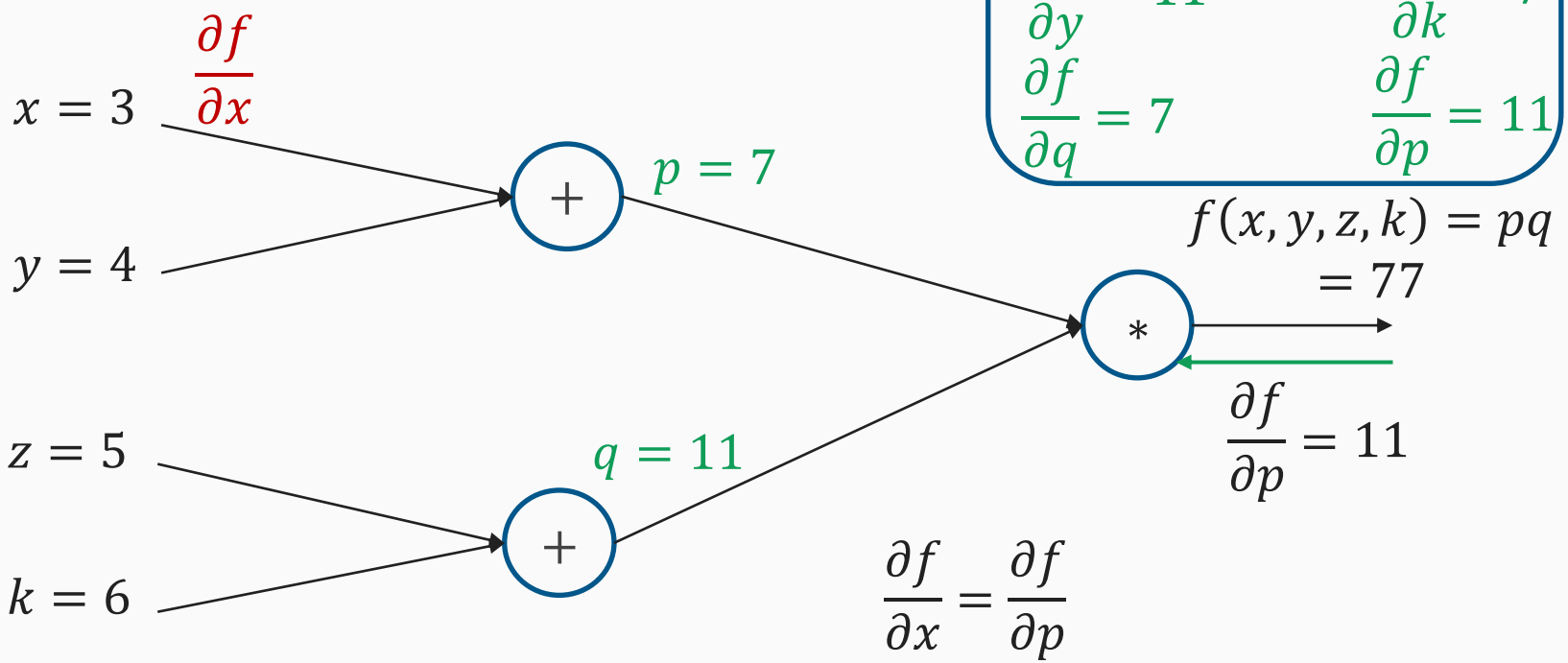
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

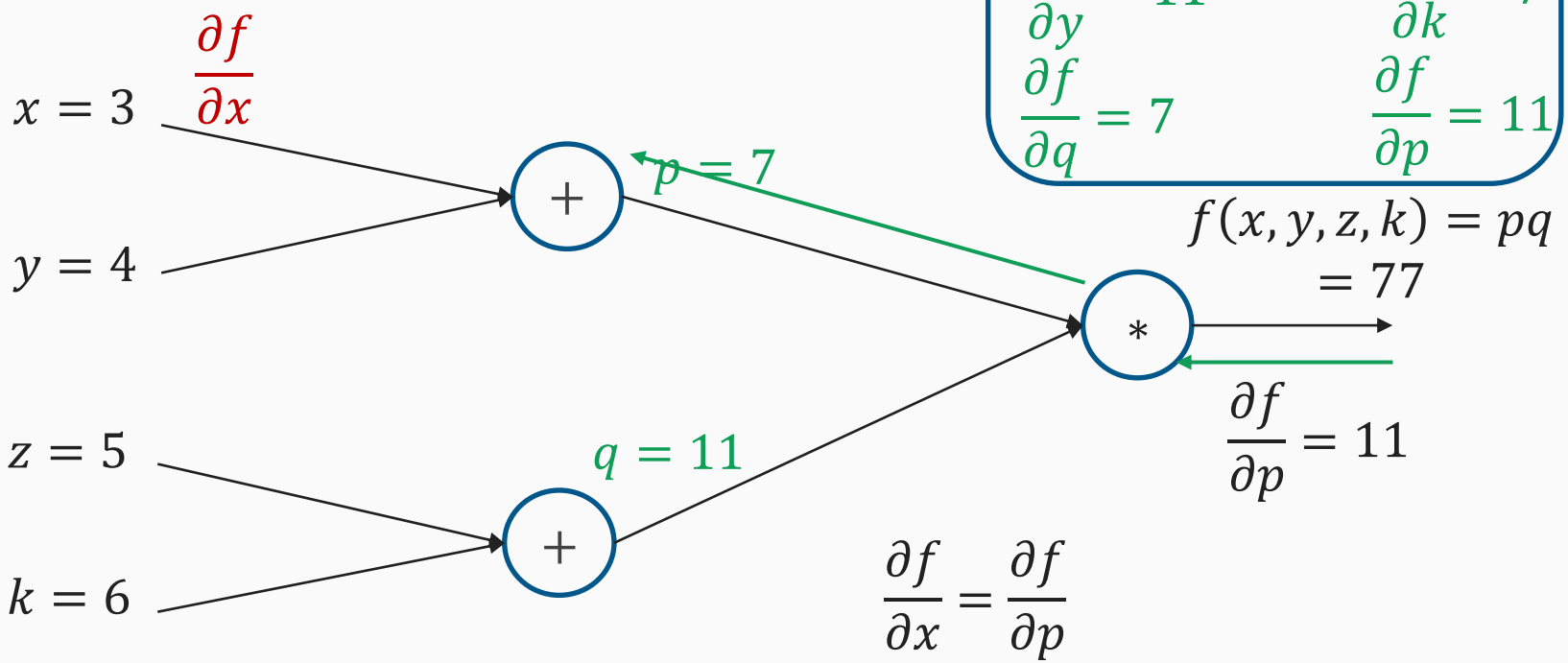
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

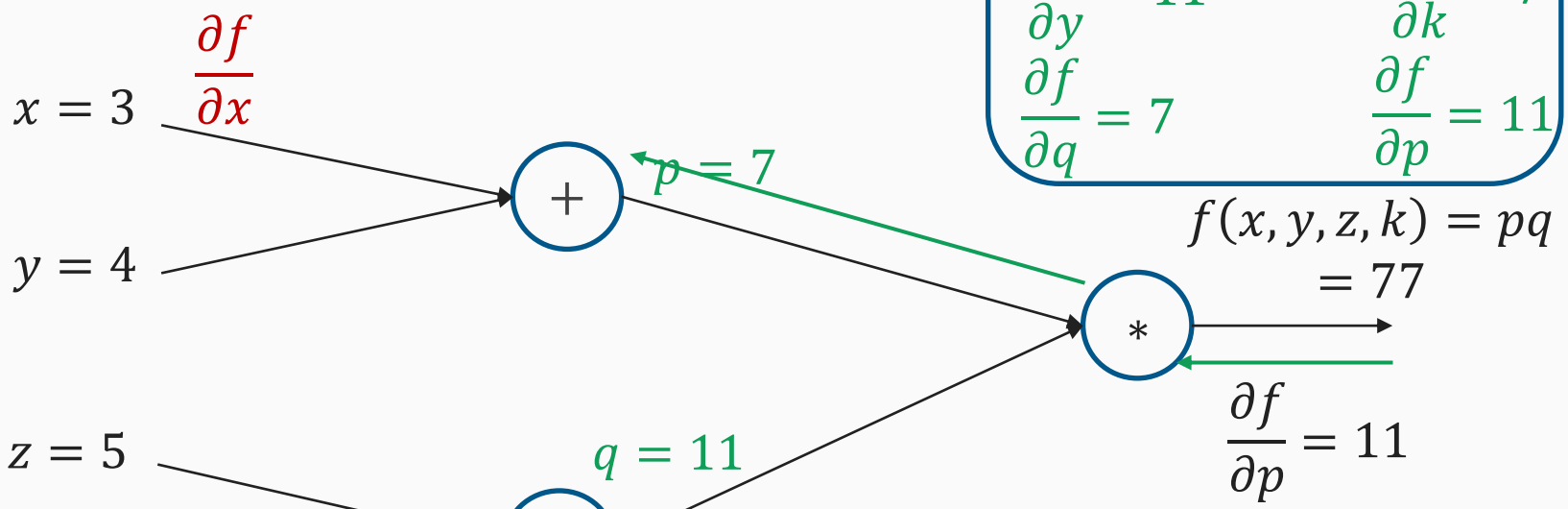
backward pass:

memory

$$\frac{\partial p}{\partial x} = 11 \qquad \frac{\partial p}{\partial z} = 7$$

$$\frac{\partial p}{\partial y} = 11 \qquad \frac{\partial p}{\partial k} = 7$$

$$\frac{\partial f}{\partial q} = 7 \qquad \frac{\partial f}{\partial p} = 11$$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

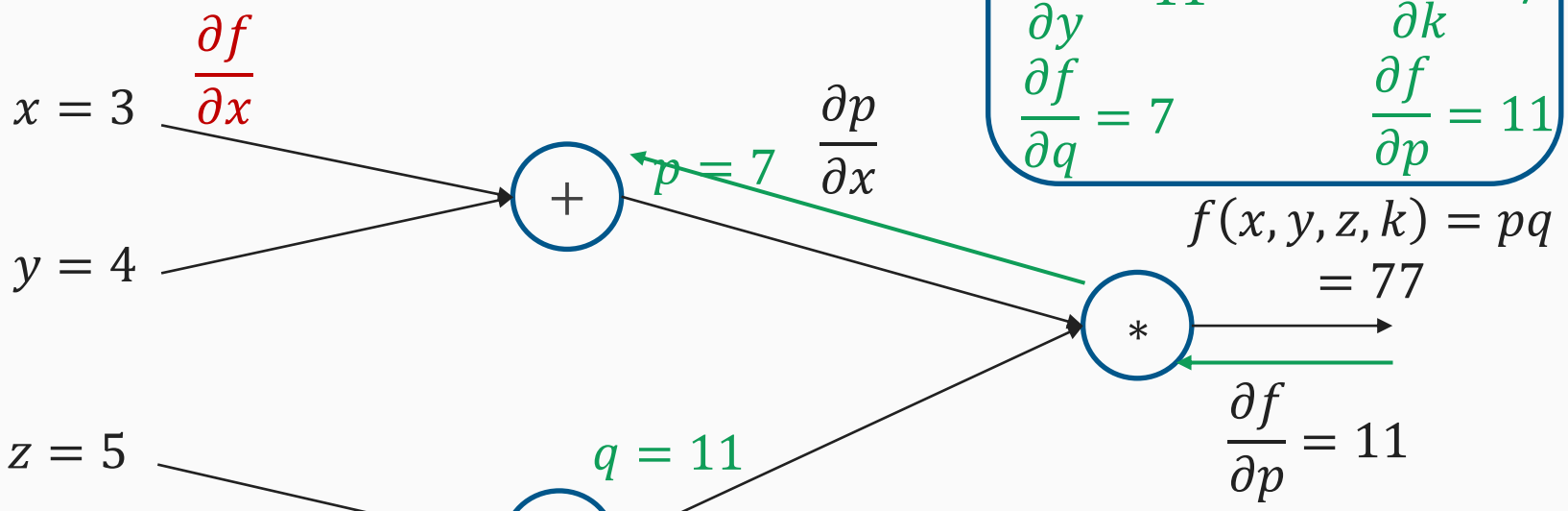
backward pass:

memory

$\frac{\partial p}{\partial x} = 11$ $\frac{\partial p}{\partial z} = 7$

$\frac{\partial p}{\partial y} = 11$ $\frac{\partial p}{\partial k} = 7$

$\frac{\partial f}{\partial q} = 7$ $\frac{\partial f}{\partial p} = 11$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

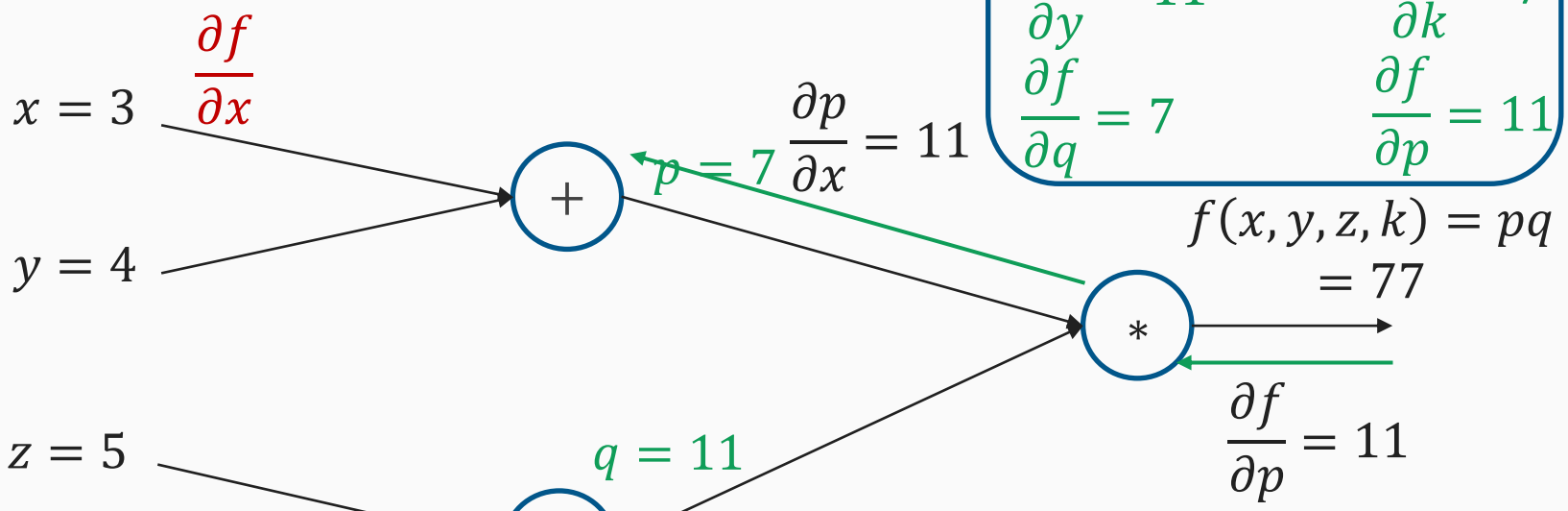
backward pass:

memory

$\frac{\partial p}{\partial x} = 11$ $\quad$ $\frac{\partial p}{\partial z} = 7$

$\frac{\partial p}{\partial y} = 11$ $\quad$ $\frac{\partial p}{\partial k} = 7$

$\frac{\partial f}{\partial q} = 7$ $\quad$ $\frac{\partial f}{\partial p} = 11$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

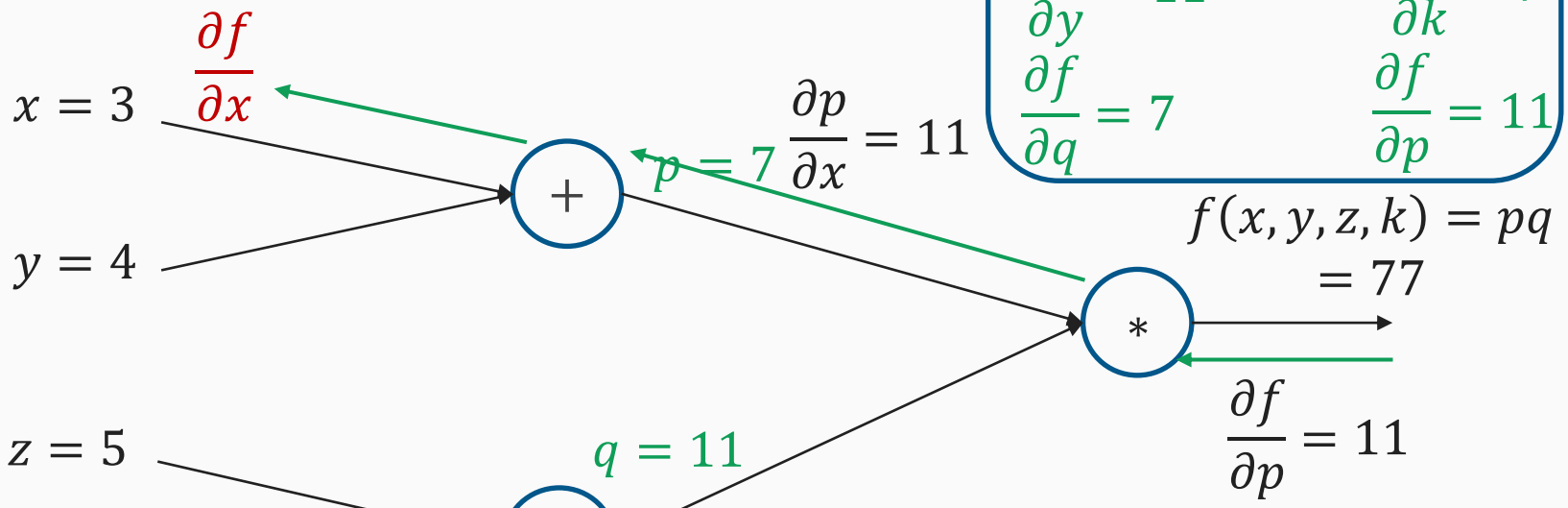
backward pass:

memory

$\frac{\partial p}{\partial x} = 11$ $\quad$ $\frac{\partial p}{\partial z} = 7$

$\frac{\partial p}{\partial y} = 11$ $\quad$ $\frac{\partial p}{\partial k} = 7$

$\frac{\partial f}{\partial q} = 7$ $\quad$ $\frac{\partial f}{\partial p} = 11$



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

$$f(x, y, z, k) = \underbrace{(x + y)}_{p}\underbrace{(z + k)}_{q}$$

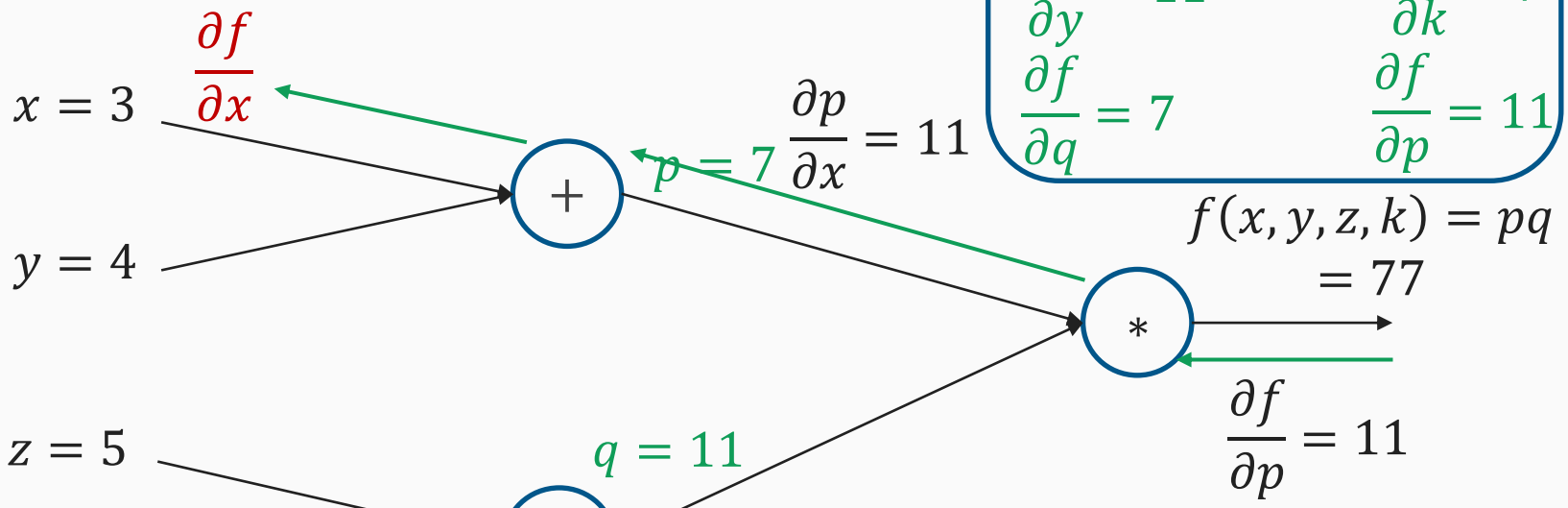
backward pass:

memory

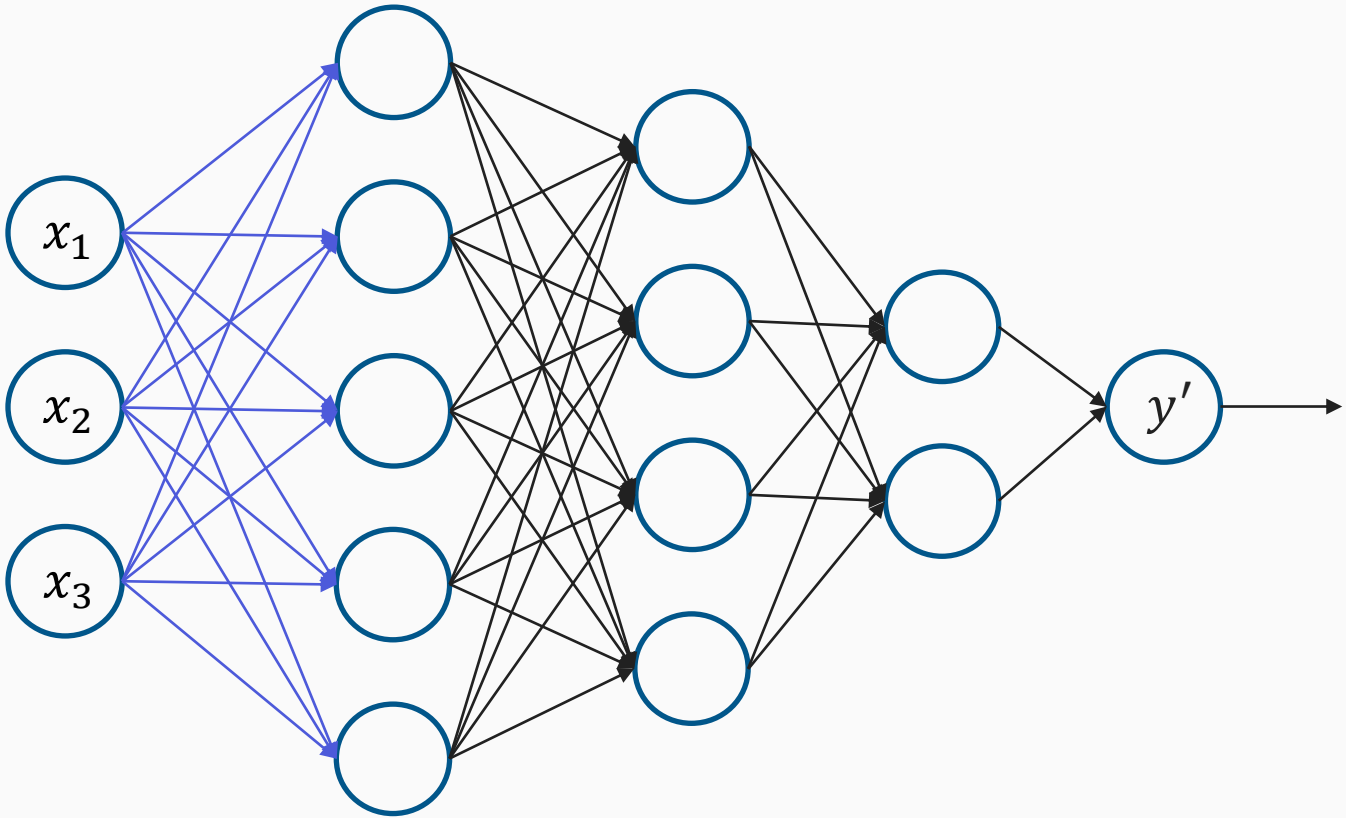
$\frac{\partial p}{\partial x} = 11$ $\quad$ $\frac{\partial p}{\partial z} = 7$

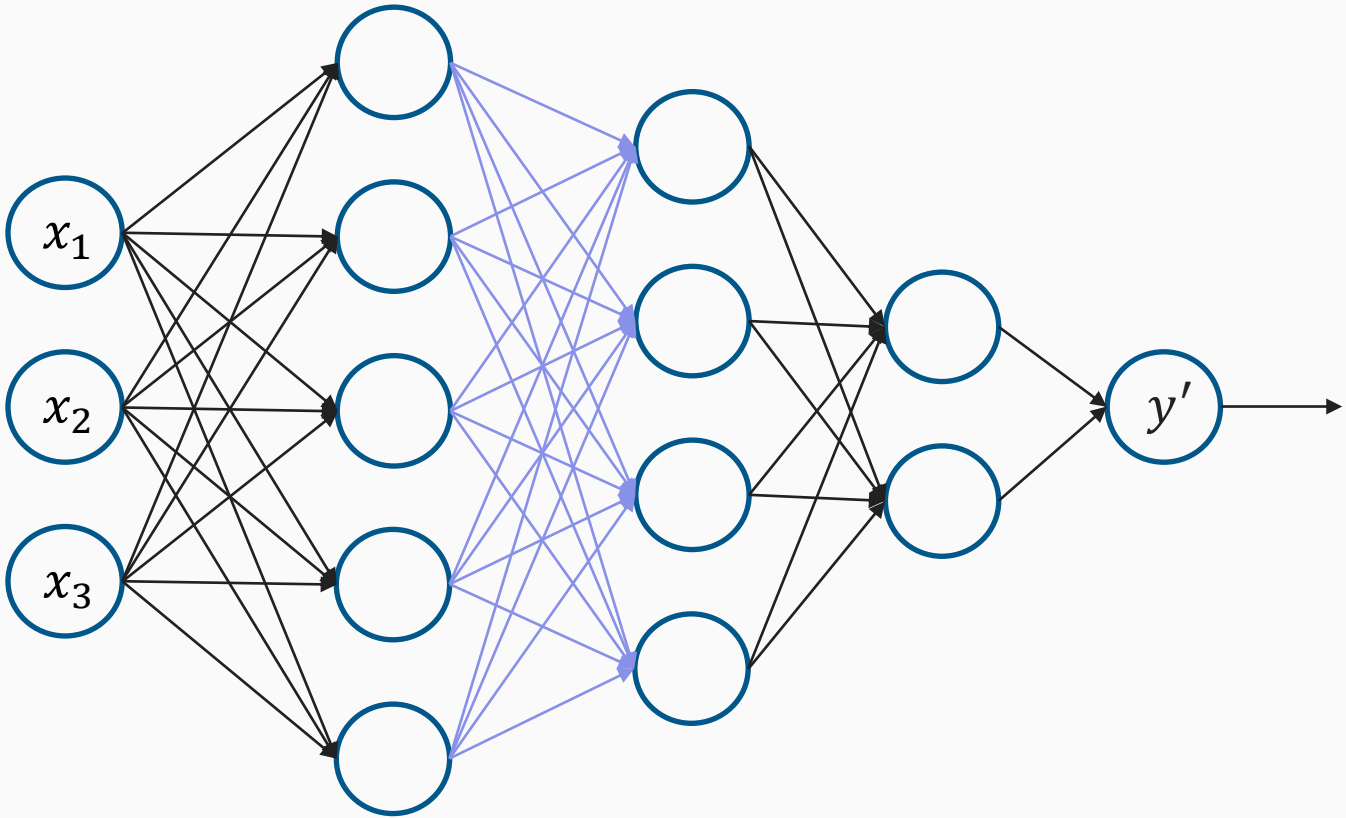
$\frac{\partial p}{\partial y} = 11$ $\quad$ $\frac{\partial p}{\partial k} = 7$

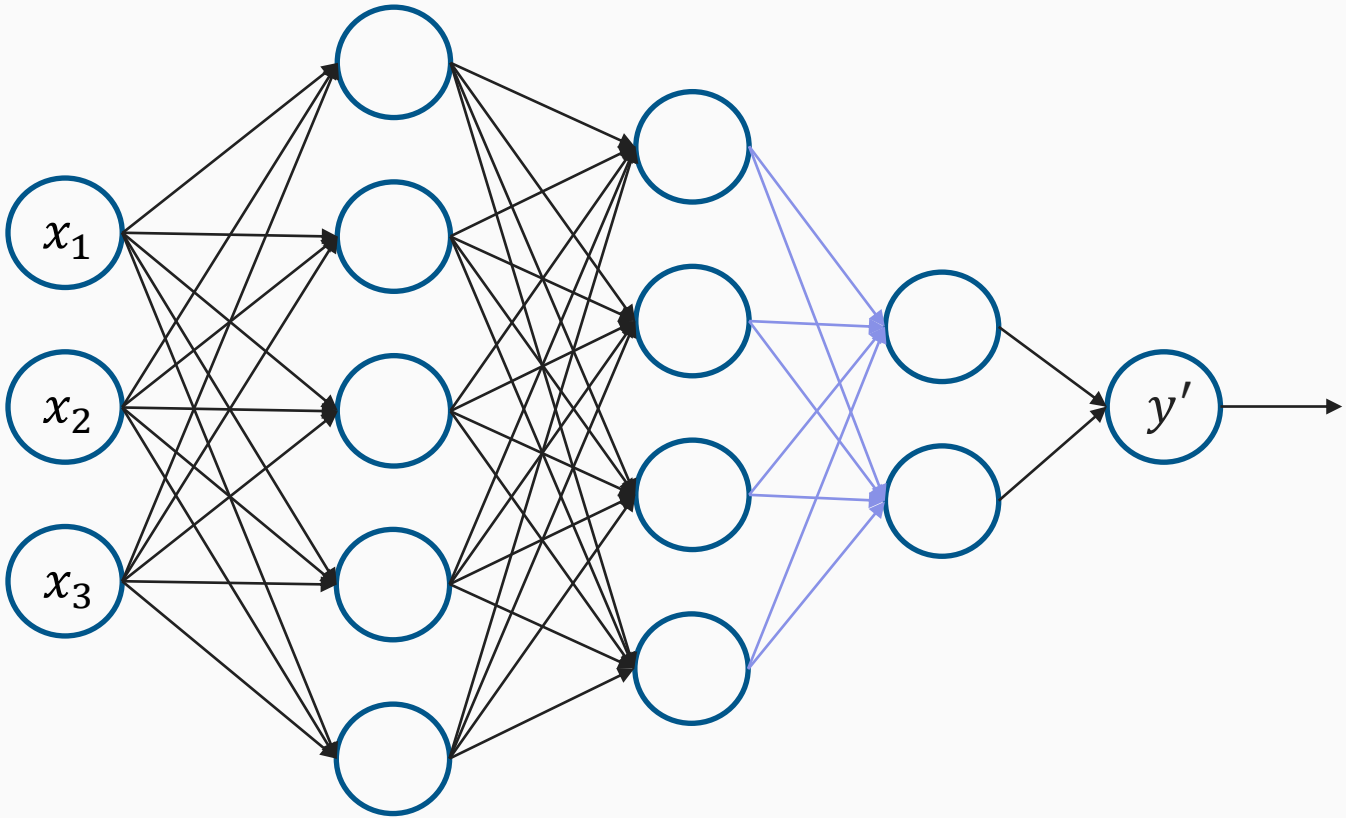
$\frac{\partial f}{\partial q} = 7$ $\quad$ $\frac{\partial f}{\partial p} = 11$

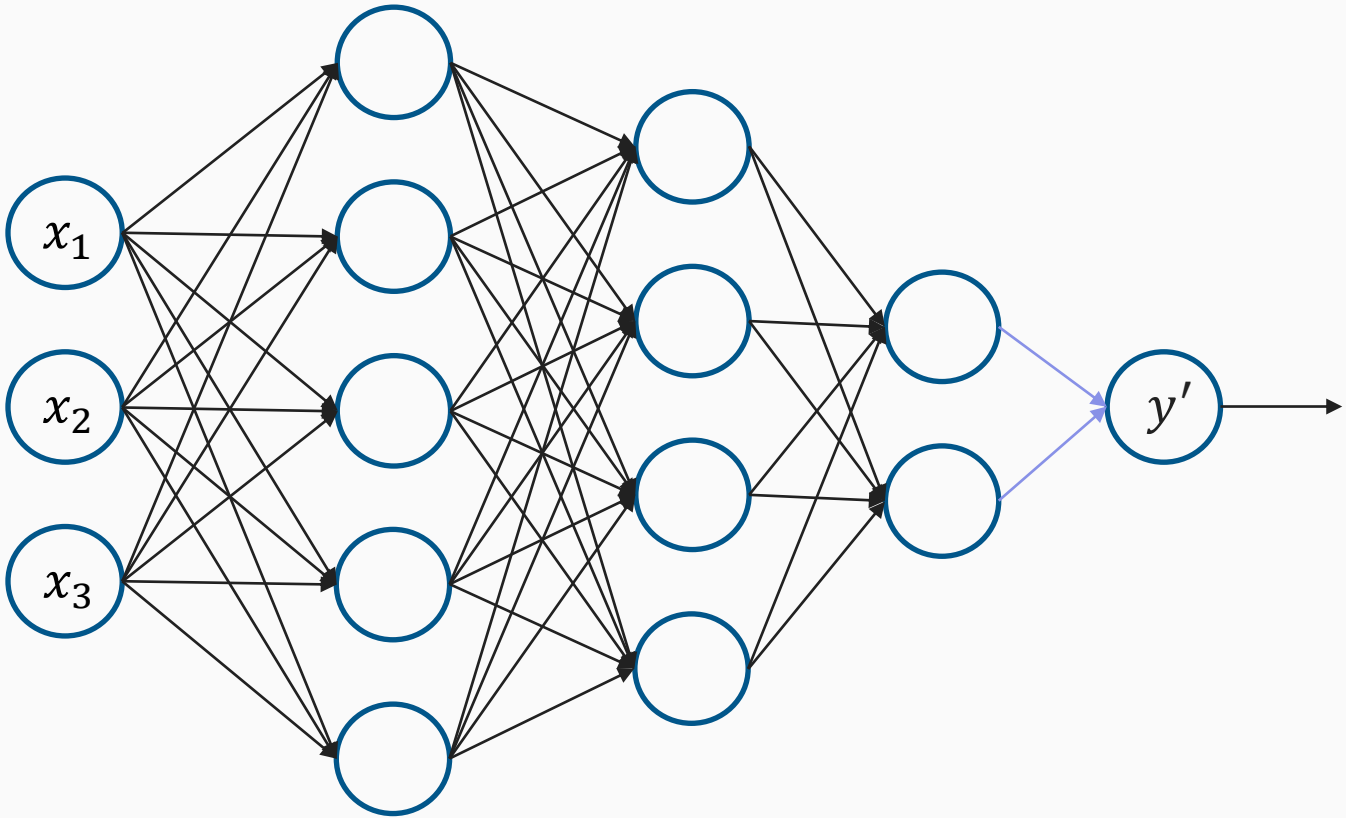


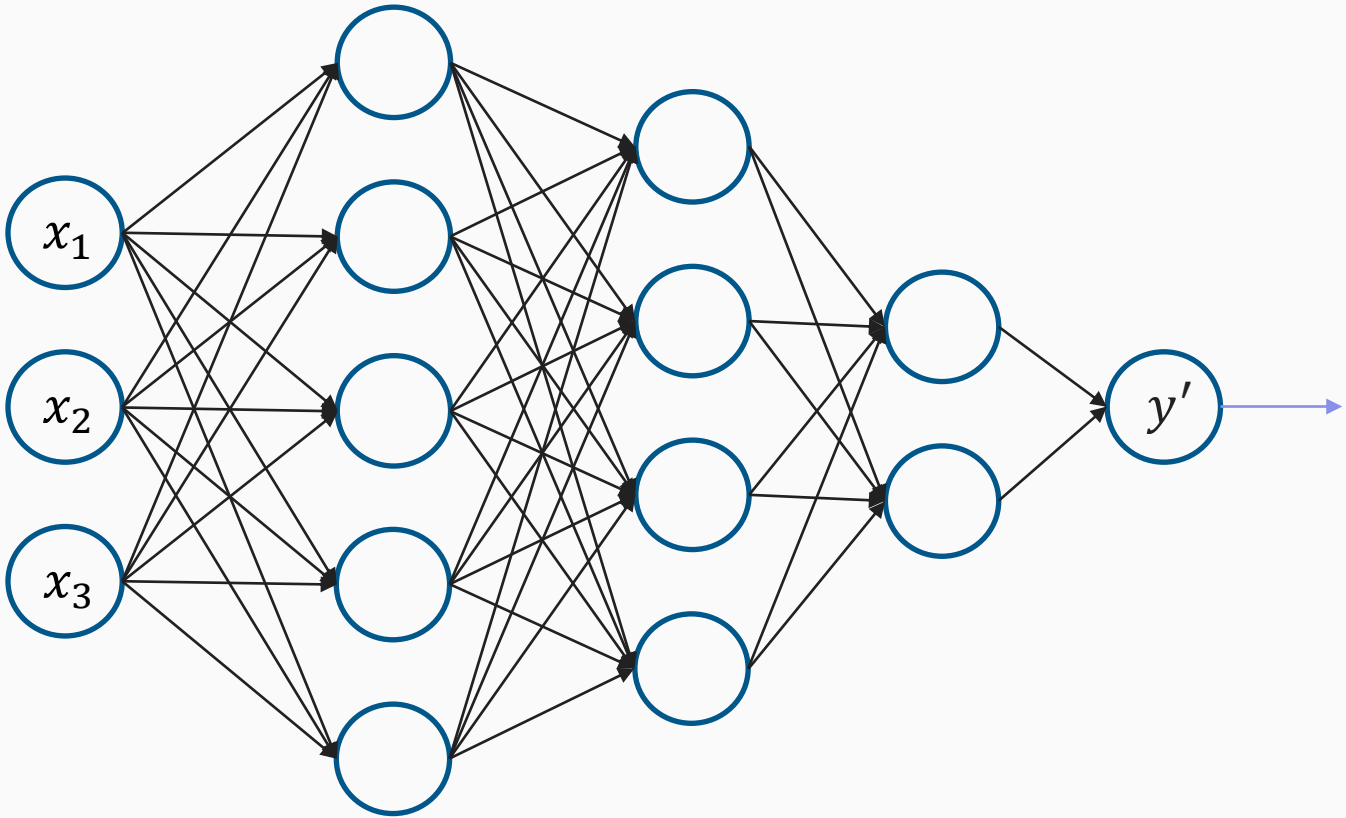
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial p}\frac{\partial p}{\partial x}$$

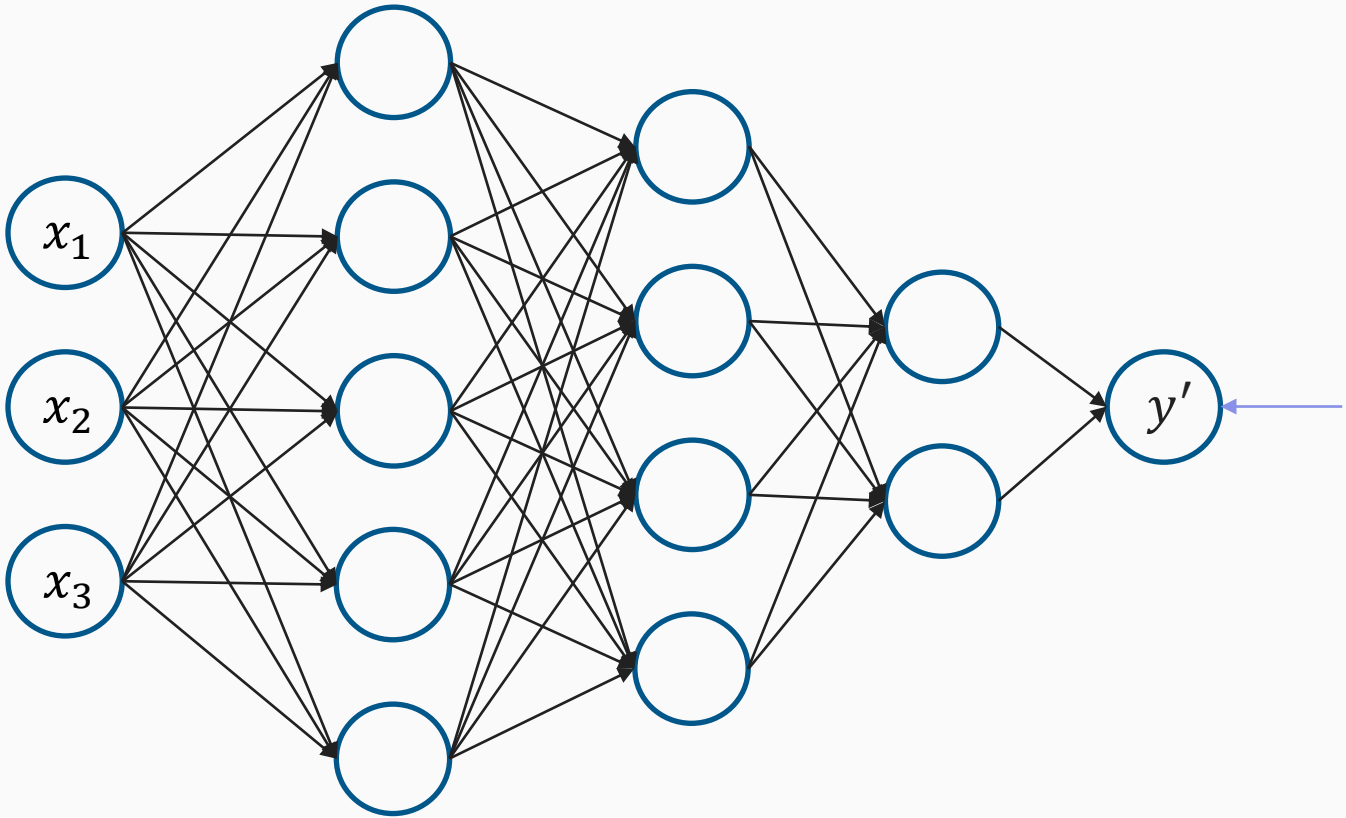

x_1
x_2
x_3
y'

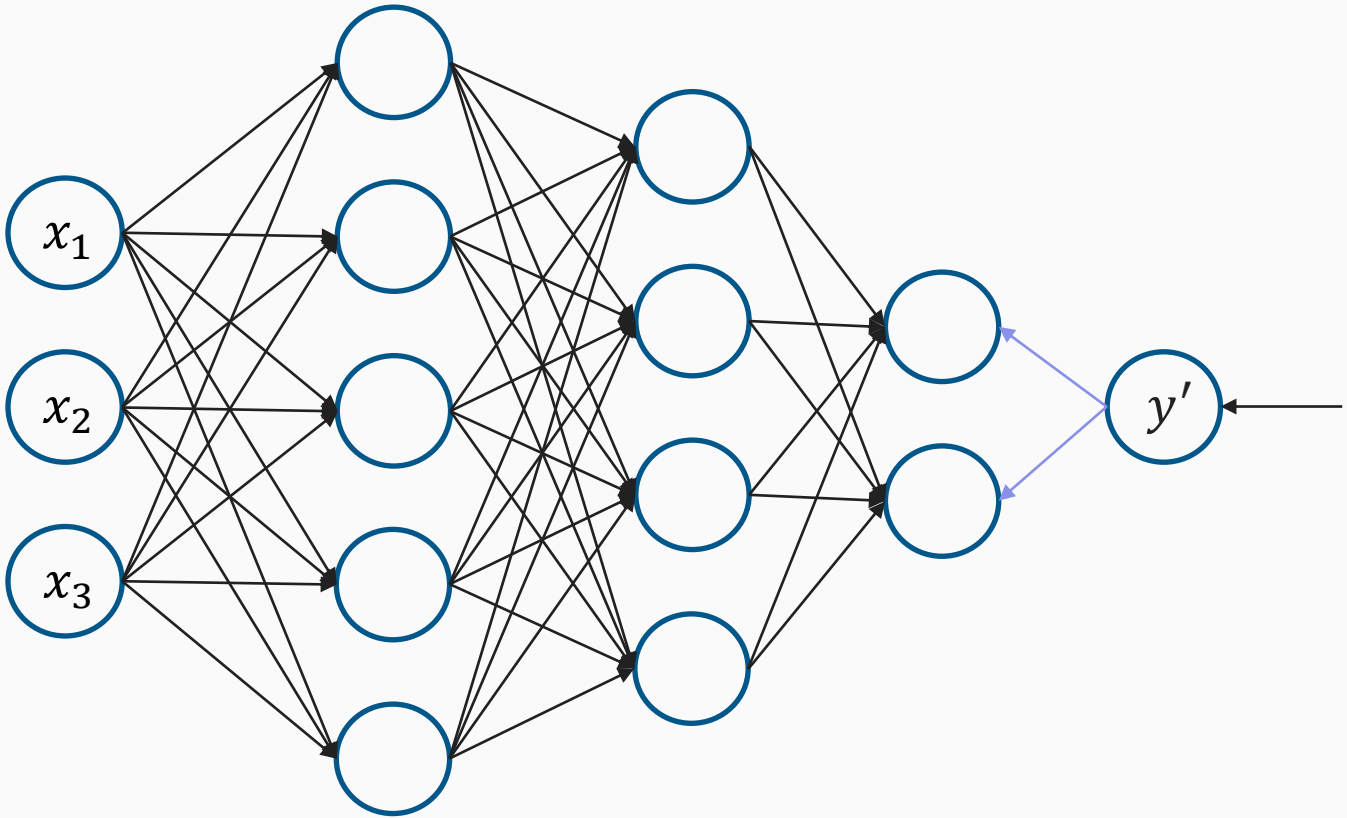

x_1
x_2
x_3
y'

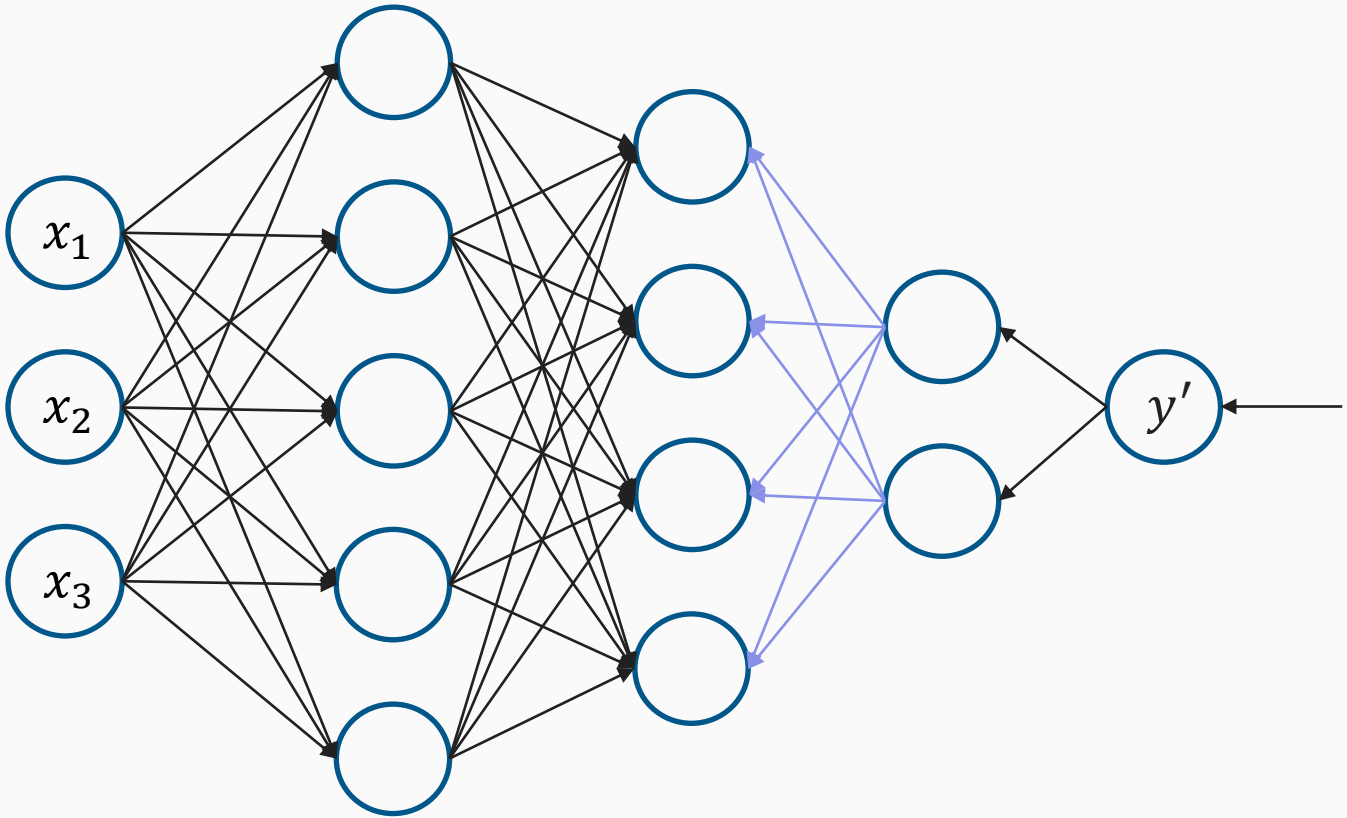

x_1
x_2
x_3
y'

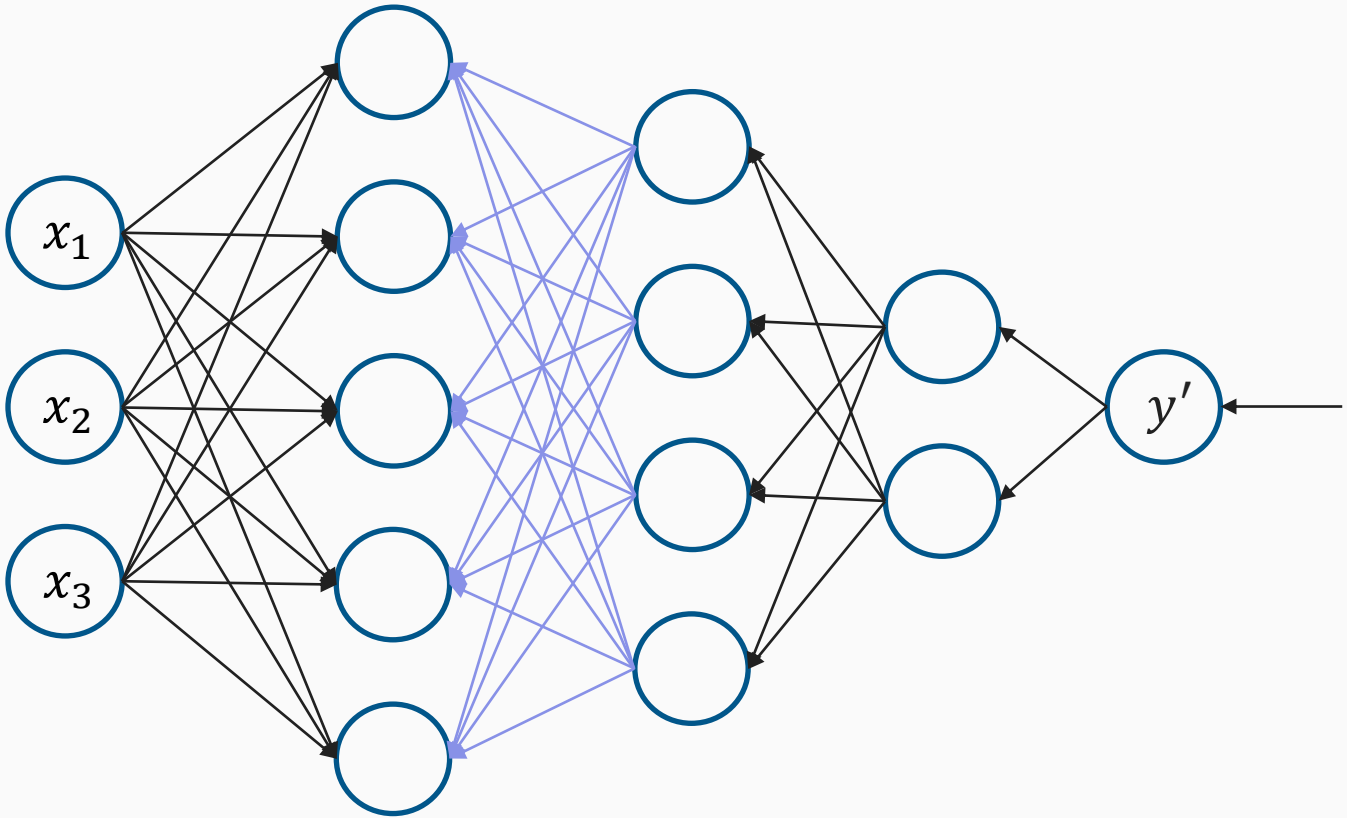

x_1
x_2
x_3
y'

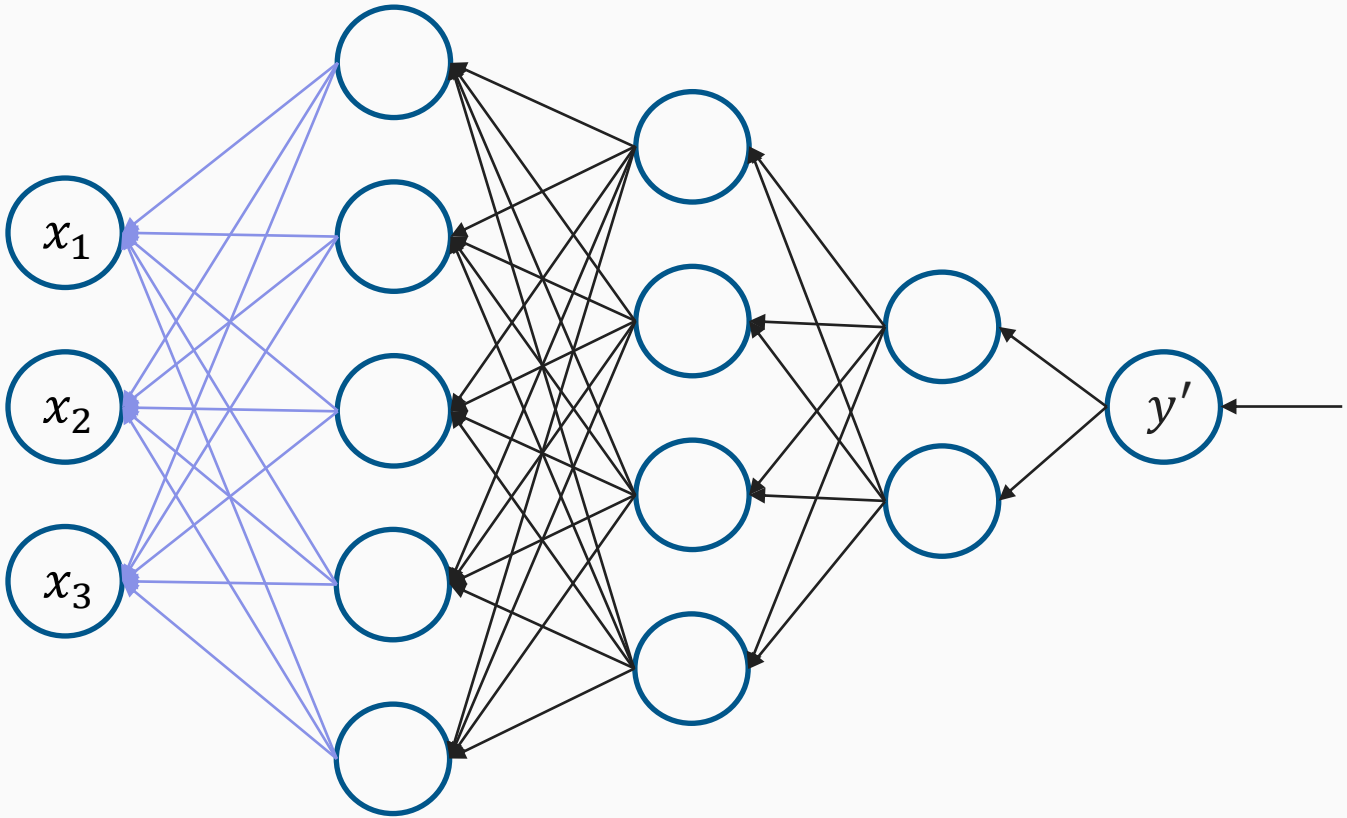

x_1
x_2
x_3
y'

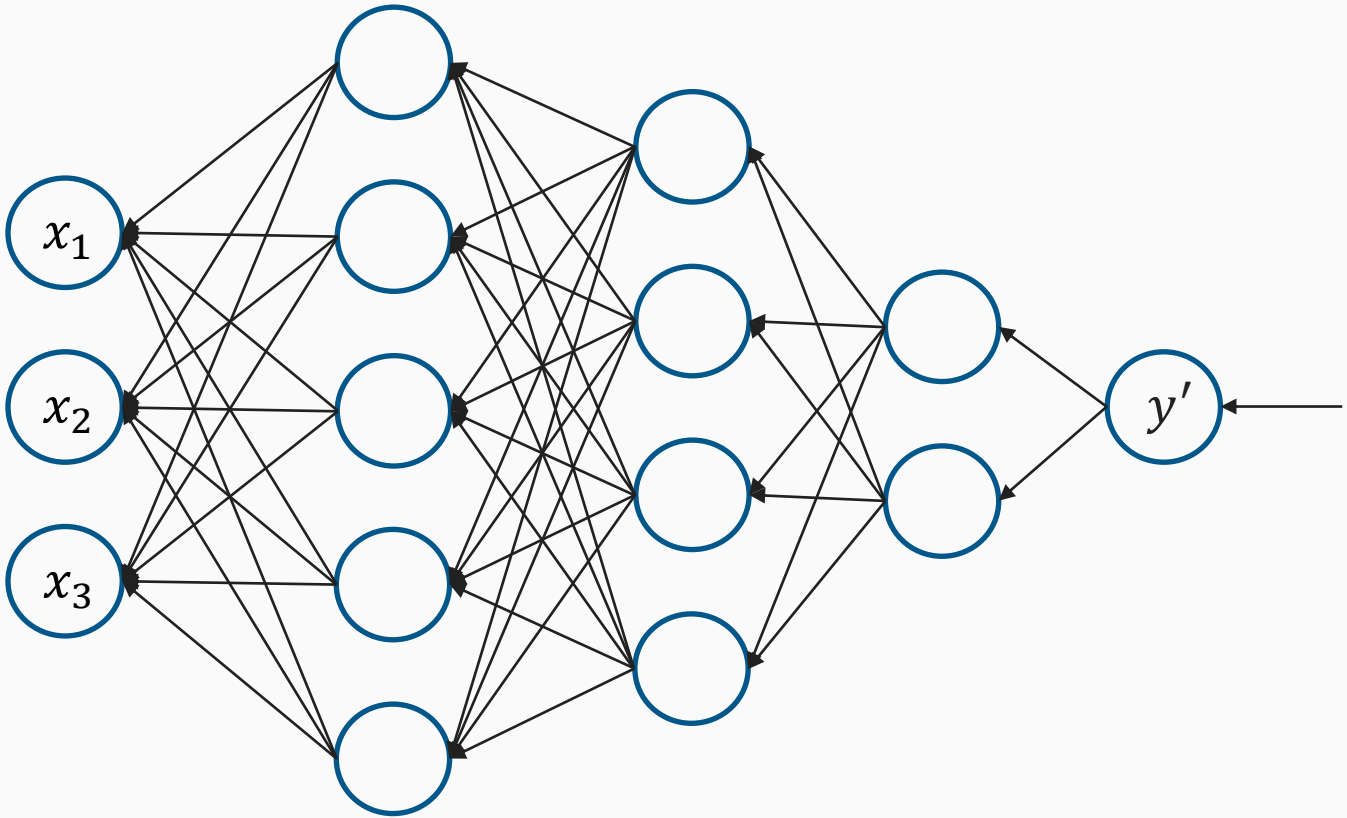

x_1
x_2
x_3
y'


x_1
x_2
x_3
y'

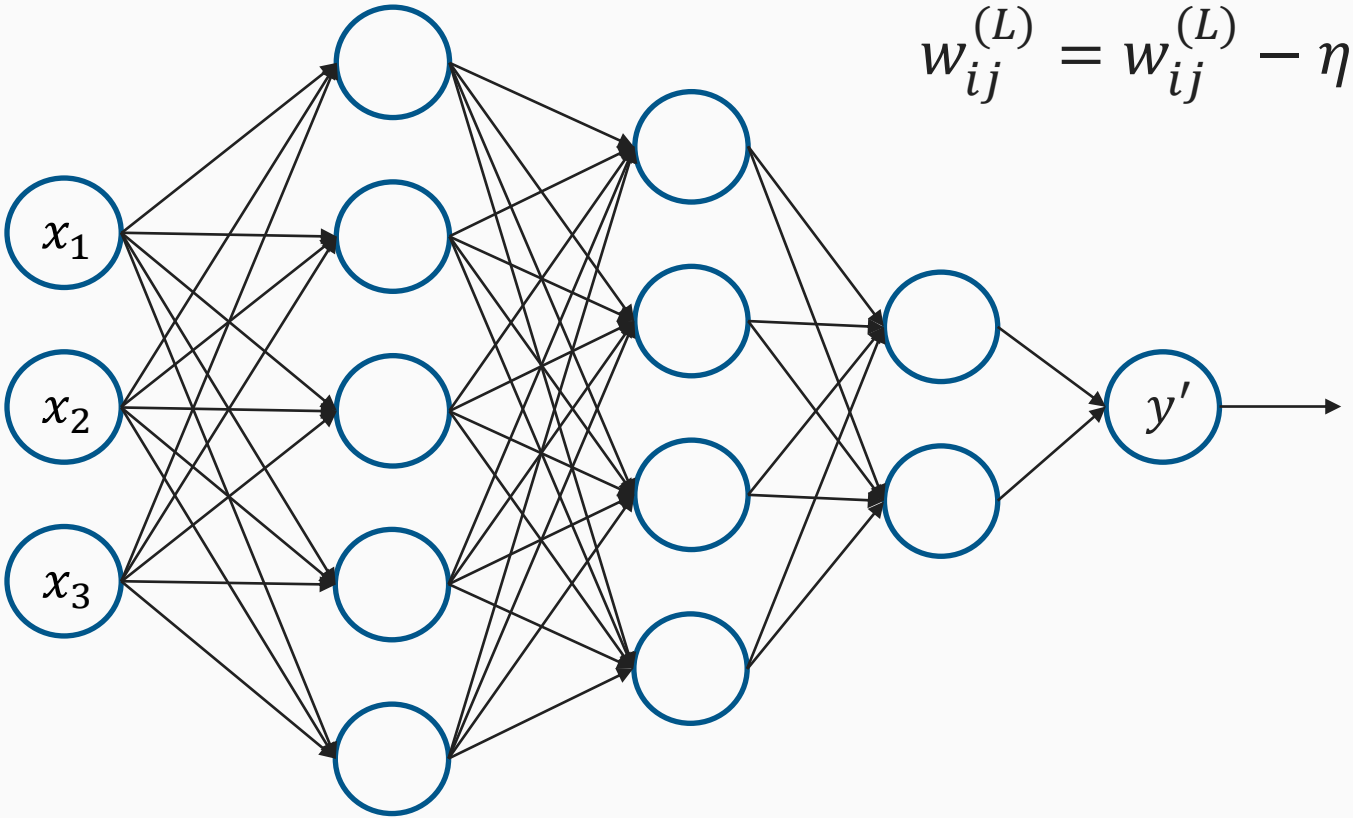

x_1
x_2
x_3
y'


x_1
x_2
x_3
y'


x_1
x_2
x_3
y'


x_1
x_2
x_3
y'

$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$



Input x

Input x $\longrightarrow$

Input x $\longrightarrow$ Hidden layers

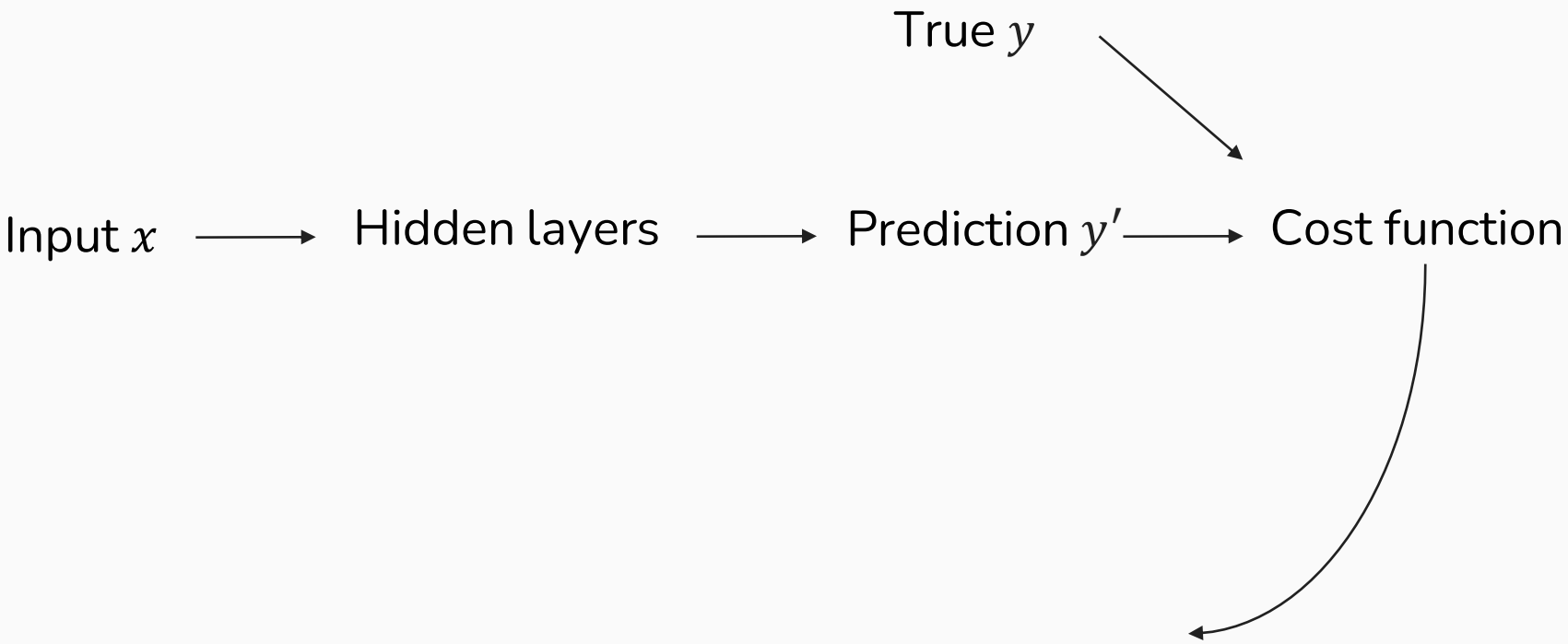
Input x $\longrightarrow$ Hidden layers $\longrightarrow$

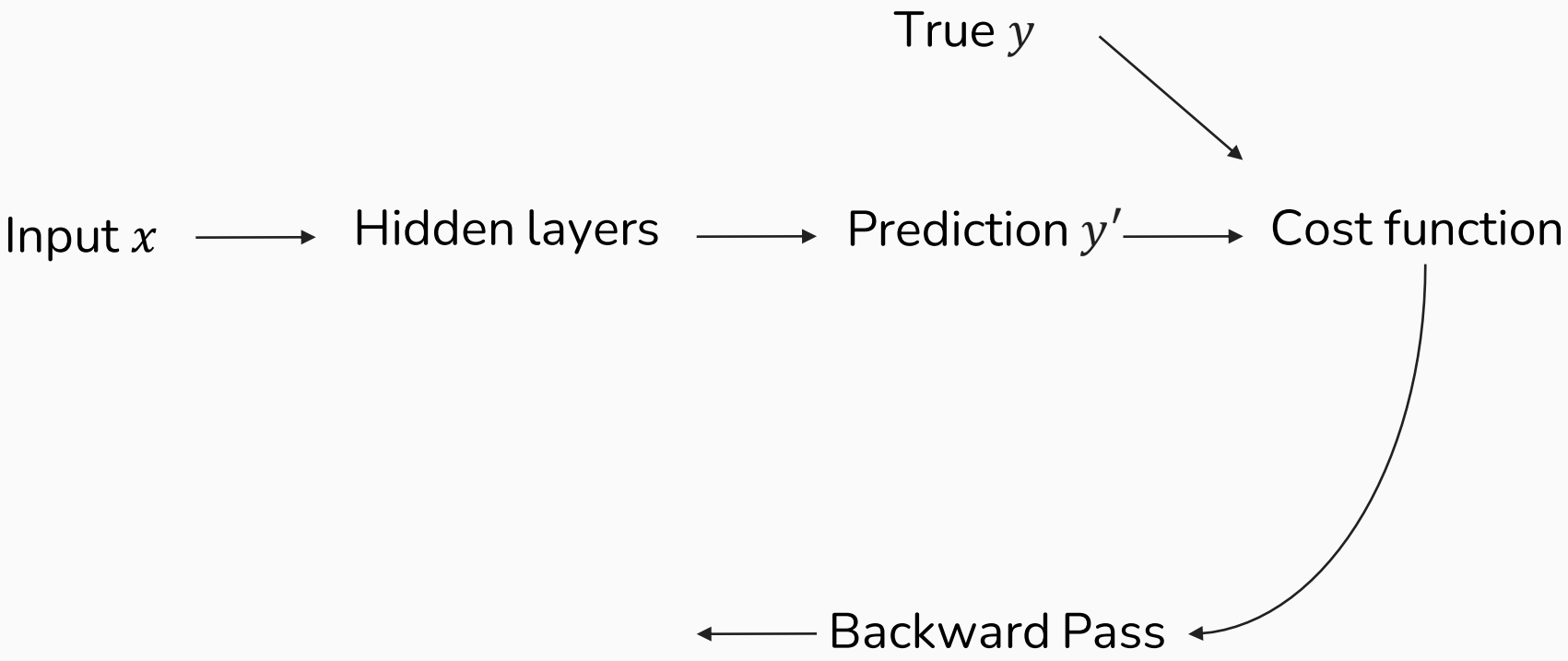
Input x $\longrightarrow$ Hidden layers $\longrightarrow$ Prediction y'

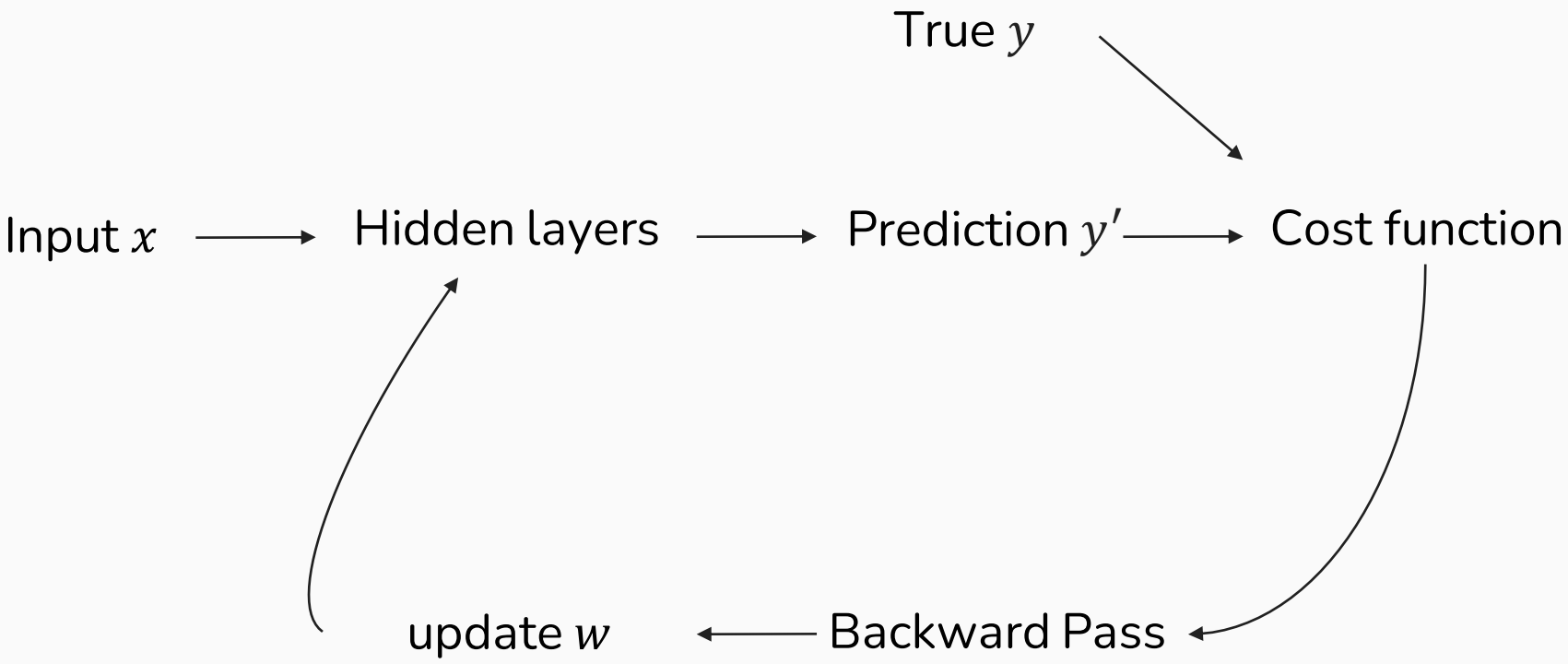
True y

Input x $\longrightarrow$ Hidden layers $\longrightarrow$ Prediction y'

$$\begin{tikzcd}
 & & \text{True } y \arrow[dr] & \\
\text{Input } x \arrow[r] & \text{Hidden layers} \arrow[r] & \text{Prediction } y' \arrow[r] & \text{Cost function}
\end{tikzcd}$$


True y
Input x
Hidden layers
Prediction y'
Cost function


True y
Input x
Hidden layers
Prediction y'
Cost function
Backward Pass


True y
Input x
Hidden layers
Prediction y'
Cost function
update w
Backward Pass

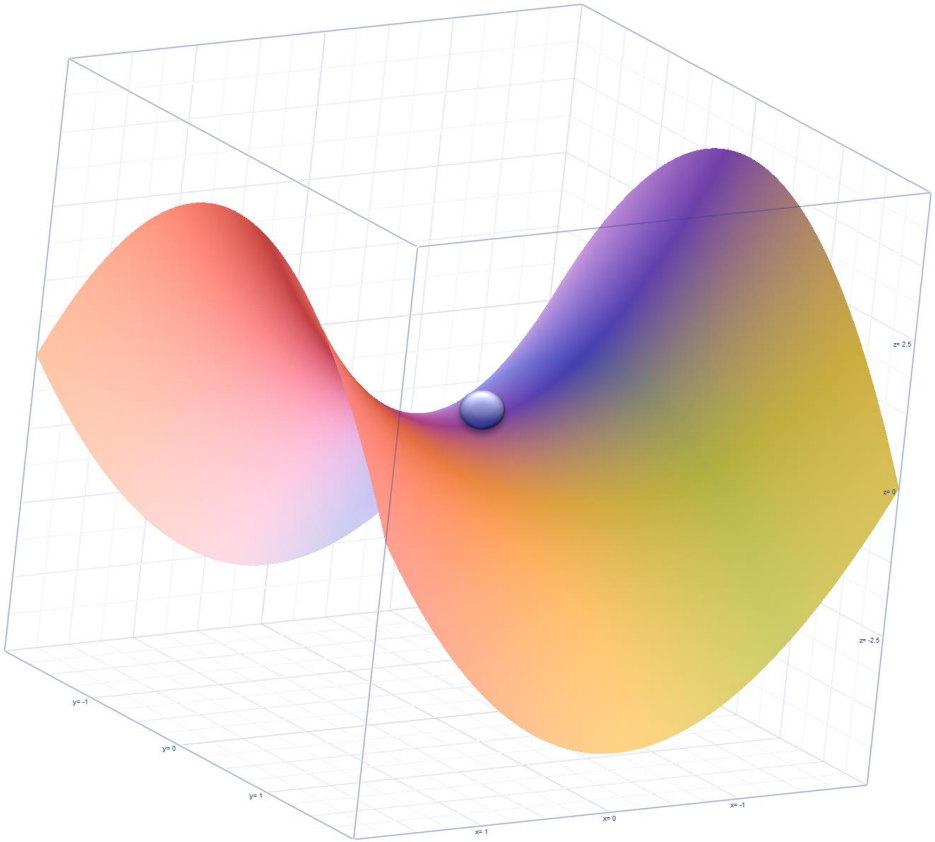
$$C'(W_3) = (O - y) \cdot R'(Z_3) \cdot H_2$$

$$C'(W_2) = (O - y) \cdot R'(Z_3) \cdot W_3 \cdot R'(Z_2) \cdot H_1$$

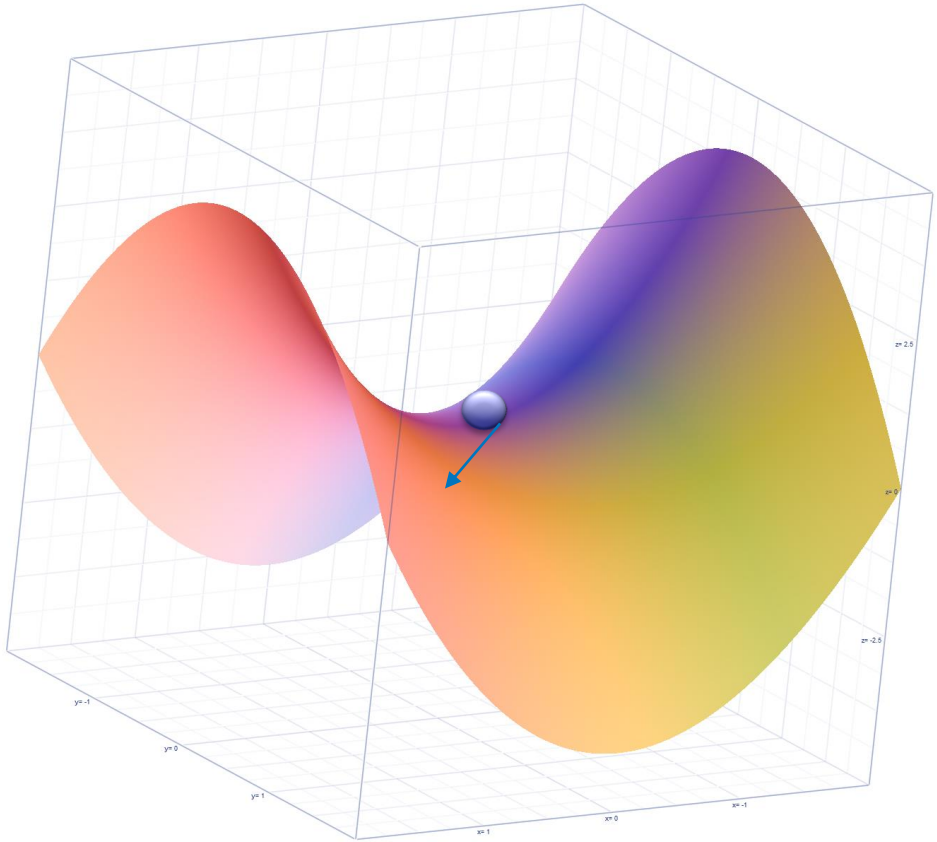
$$C'(W_1) = (O - y) \cdot R'(Z_3) \cdot W_3 \cdot R'(Z_2) \cdot W_2 \cdot R'(Z_1) \cdot H_0$$

$$C'(W_0) = (O - y) \cdot R'(Z_3) \cdot W_3 \cdot R'(Z_2) \cdot W_2 \cdot R'(Z_1) \cdot W_1 \cdot R'(Z_0) \cdot X$$

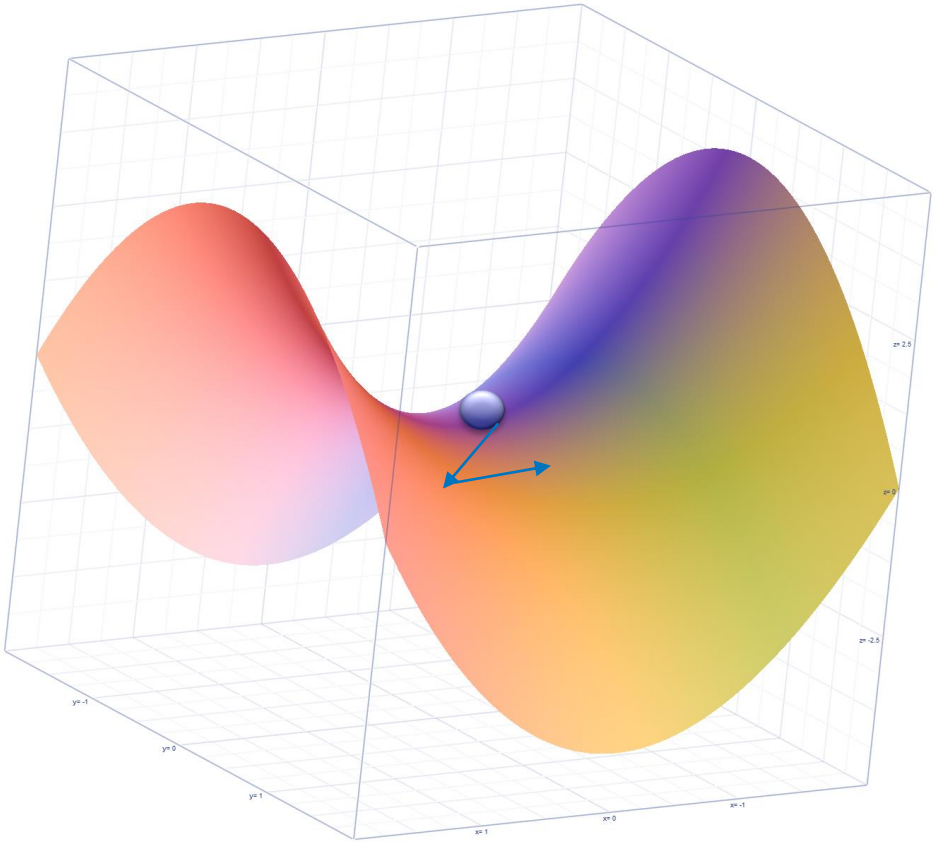
Optimizers



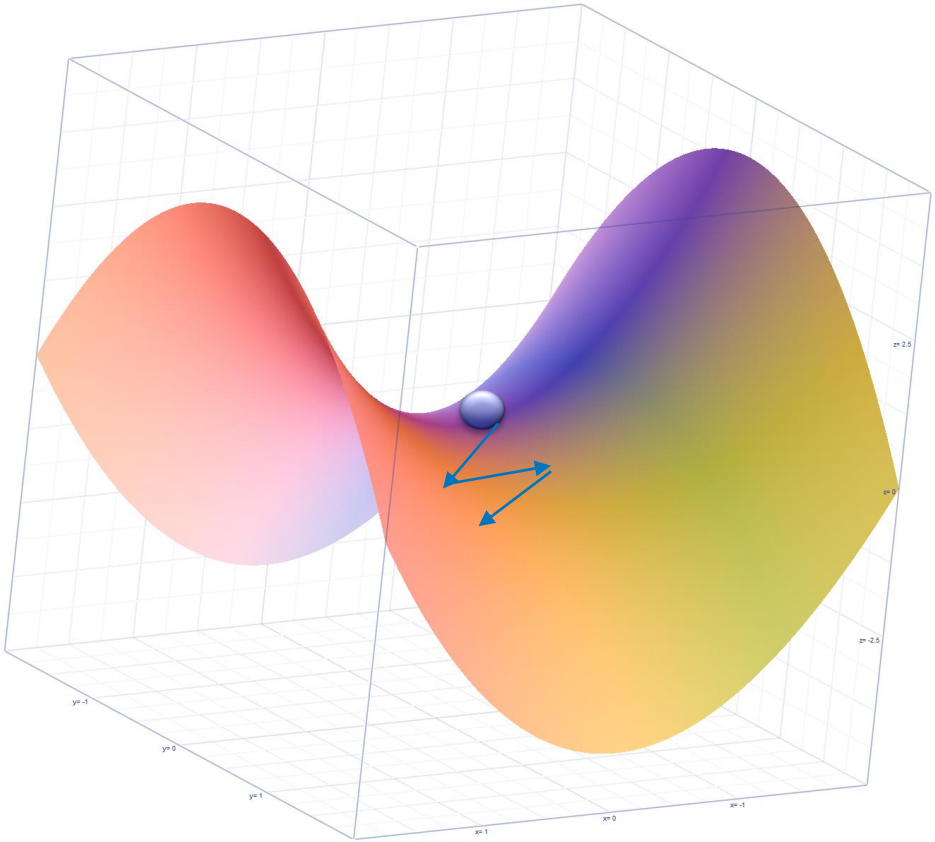
Without momentum



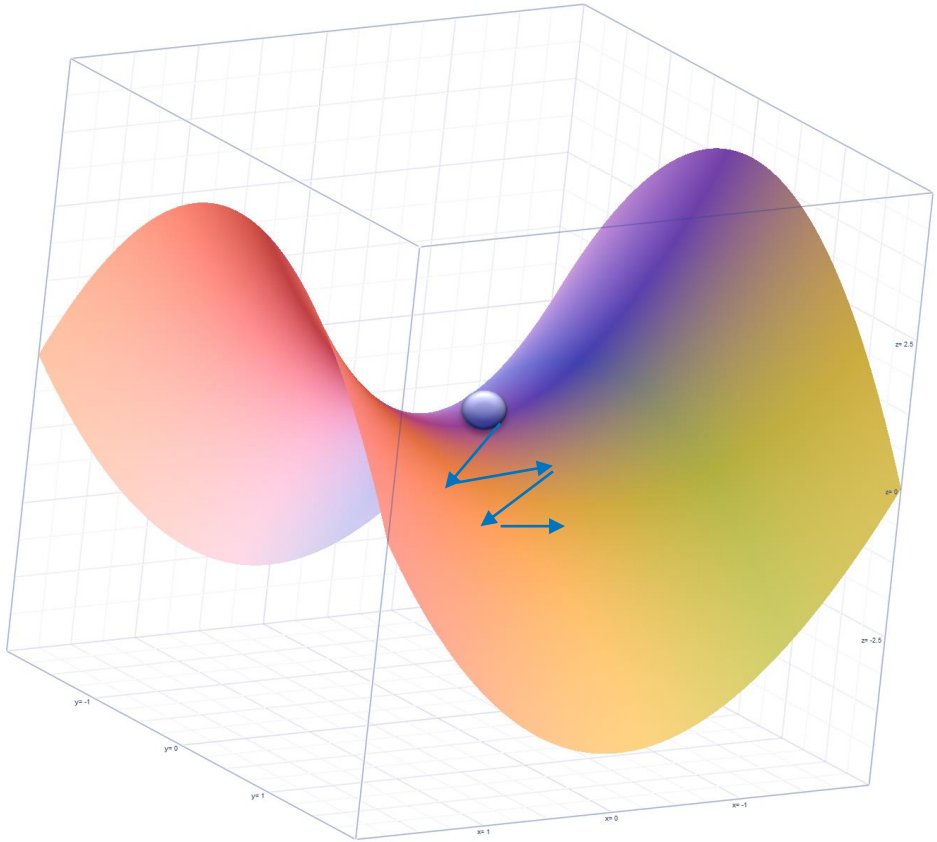
Without momentum



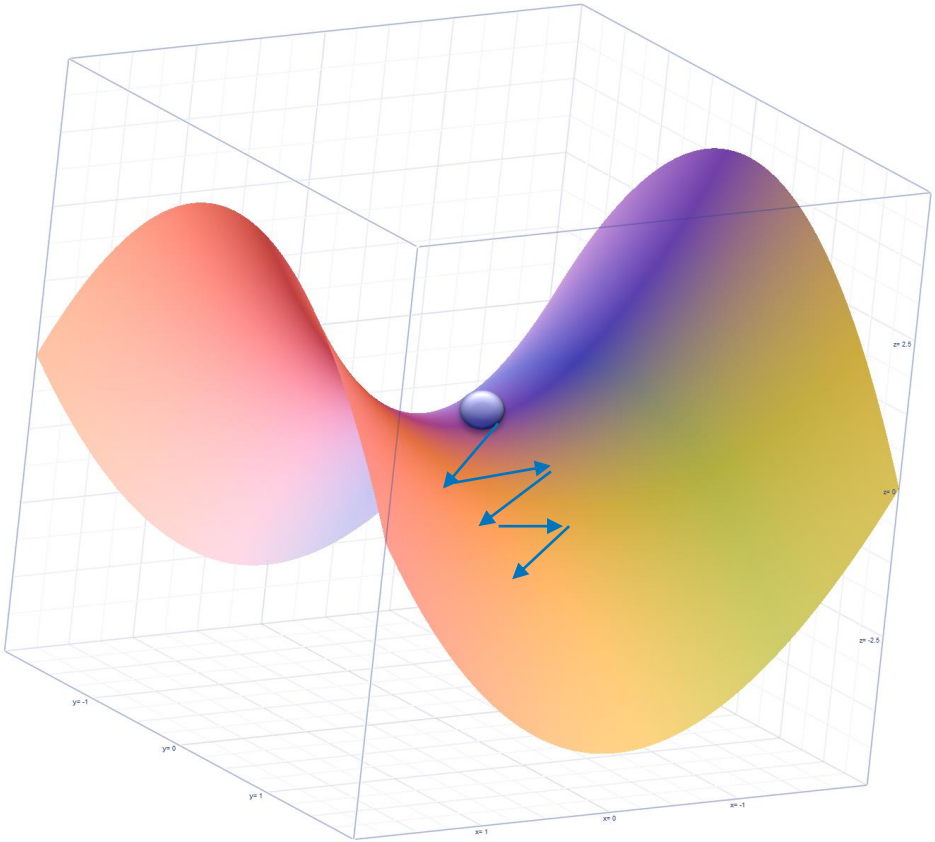
Without momentum



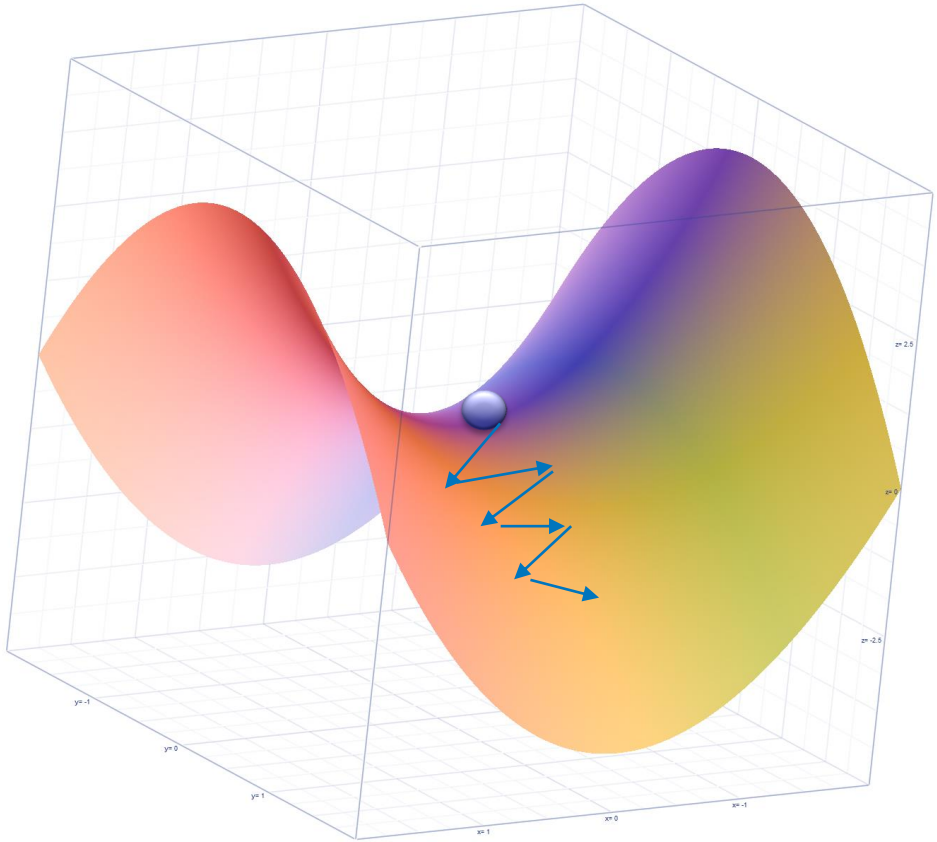
Without momentum



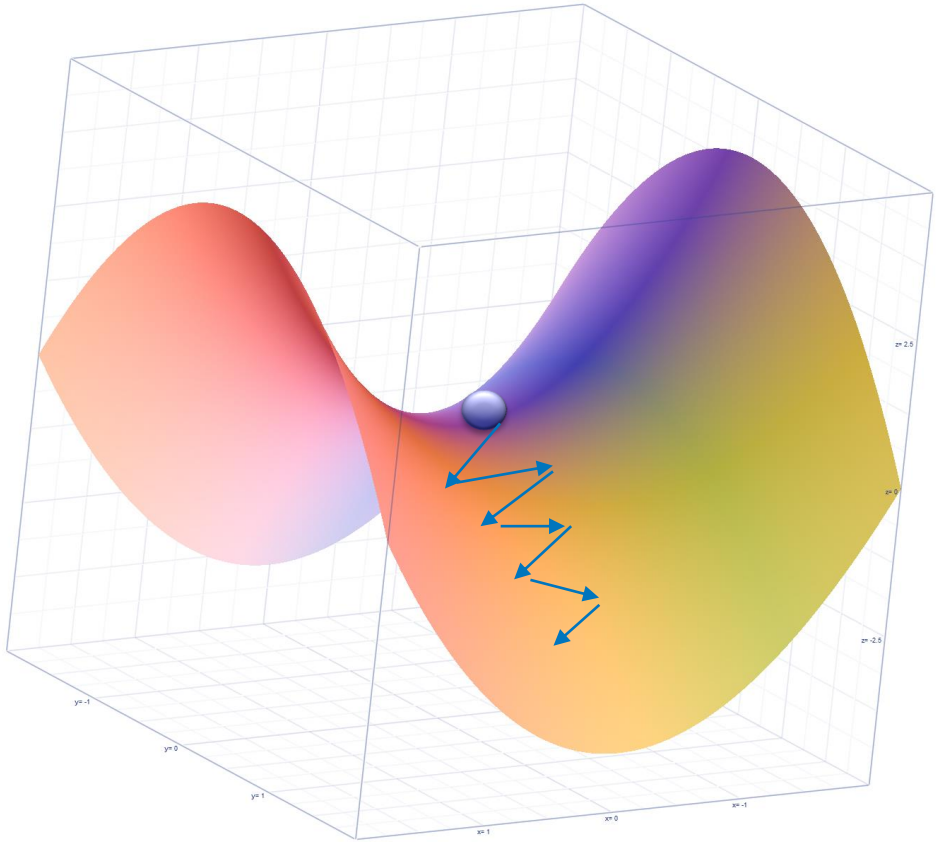
Without momentum



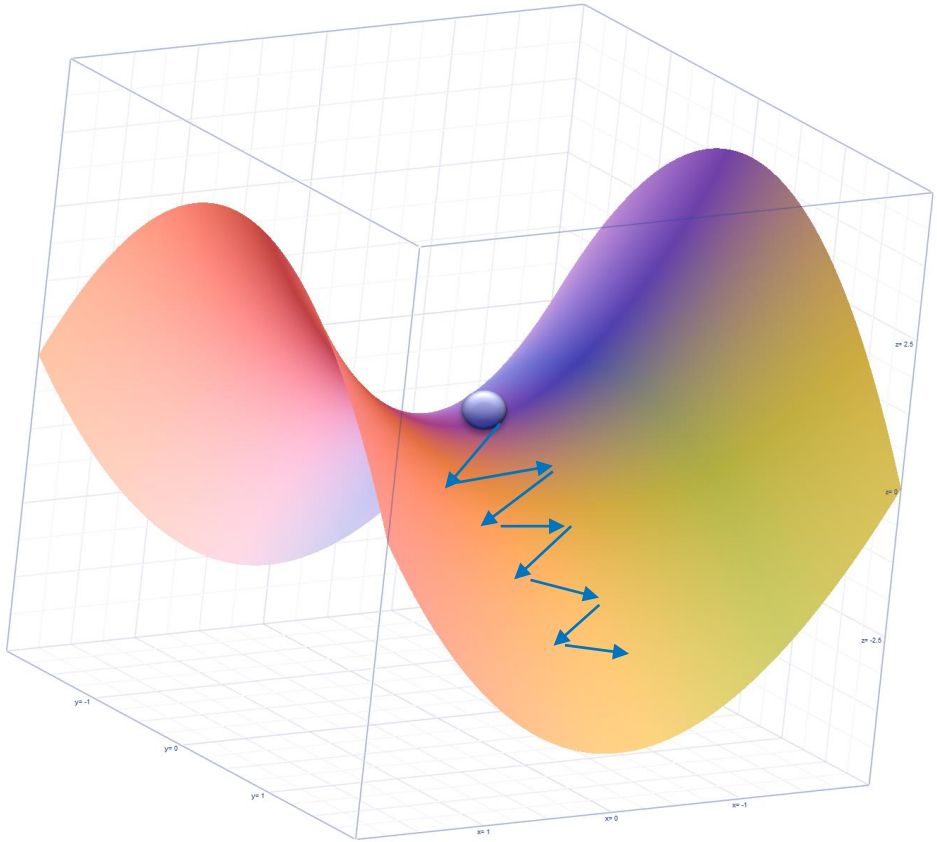
Without momentum



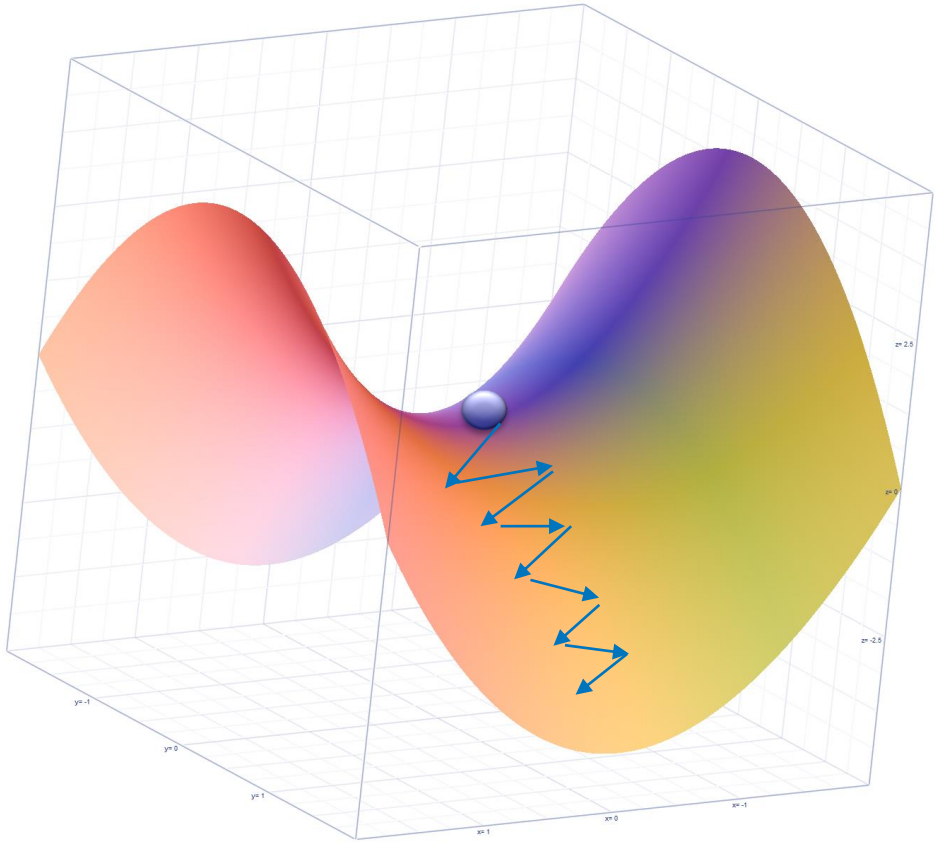
Without momentum



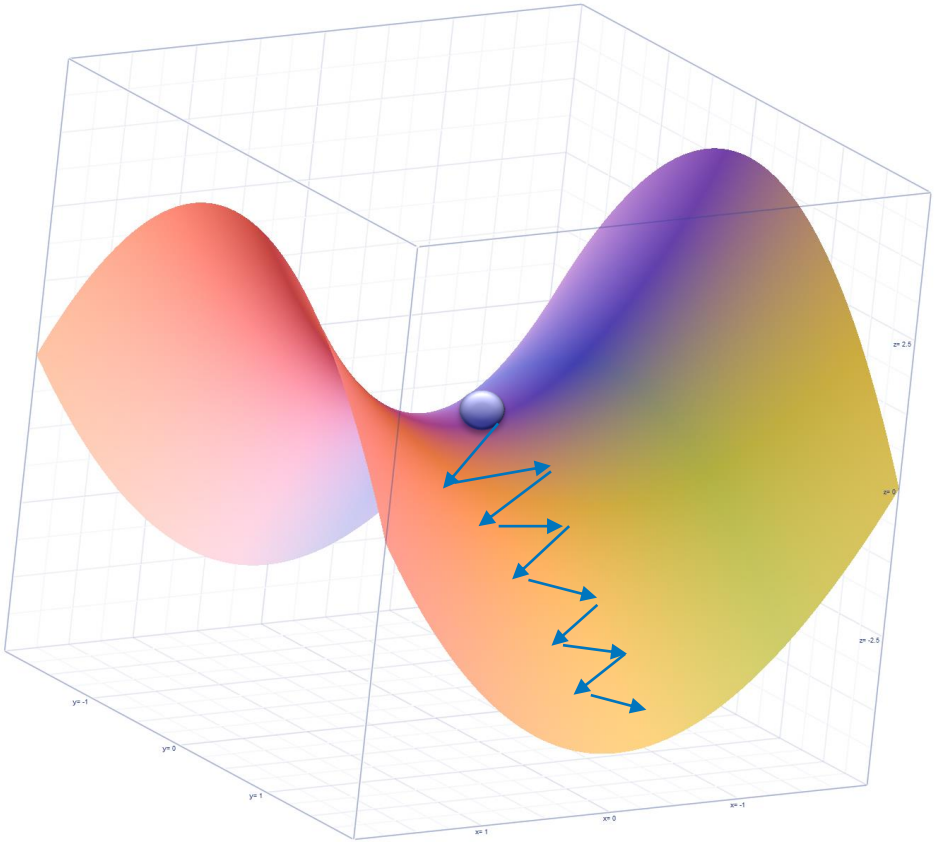
Without momentum



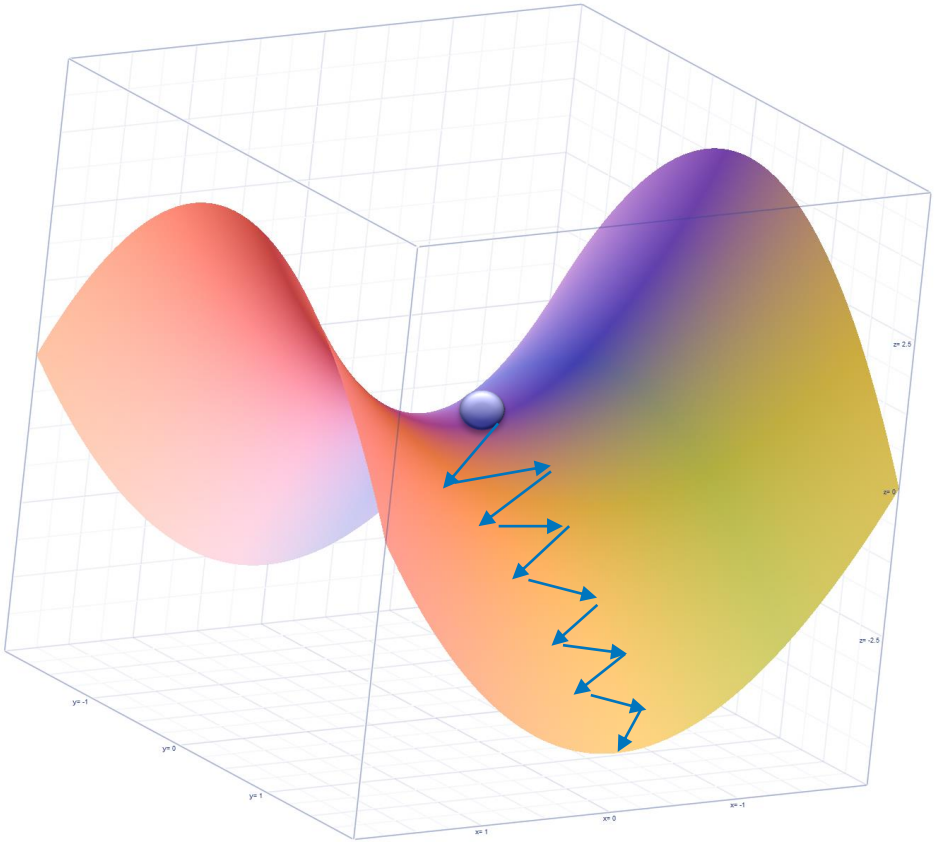
Without momentum



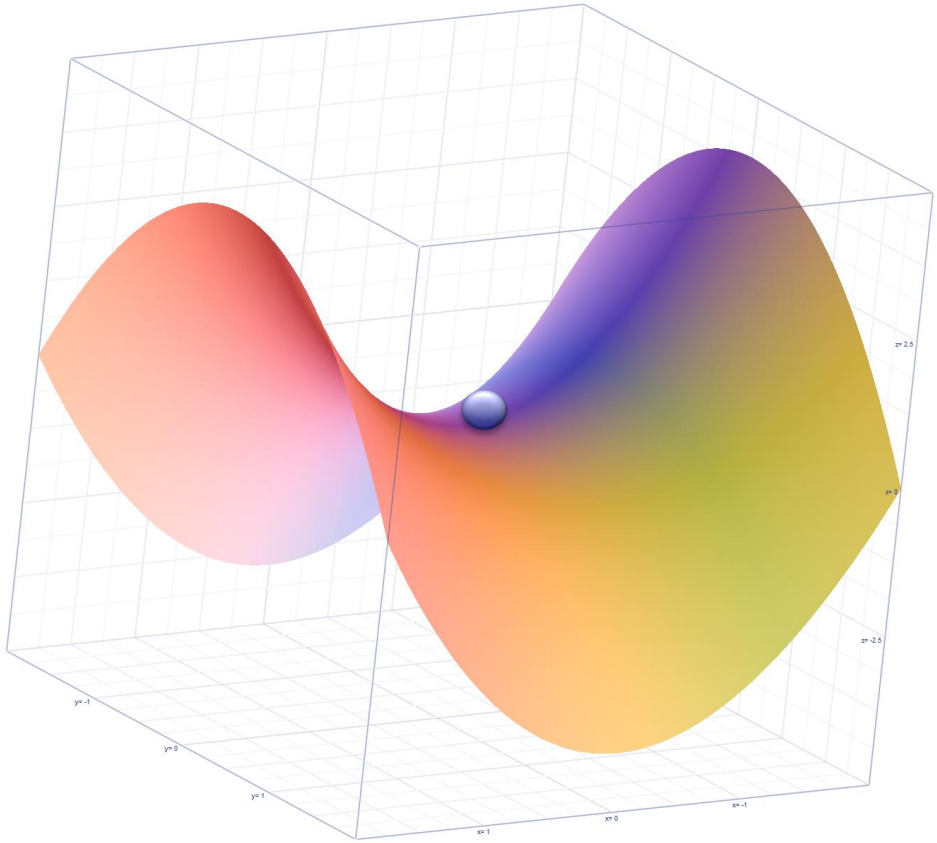
Without momentum

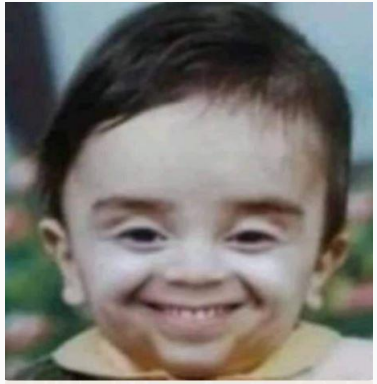


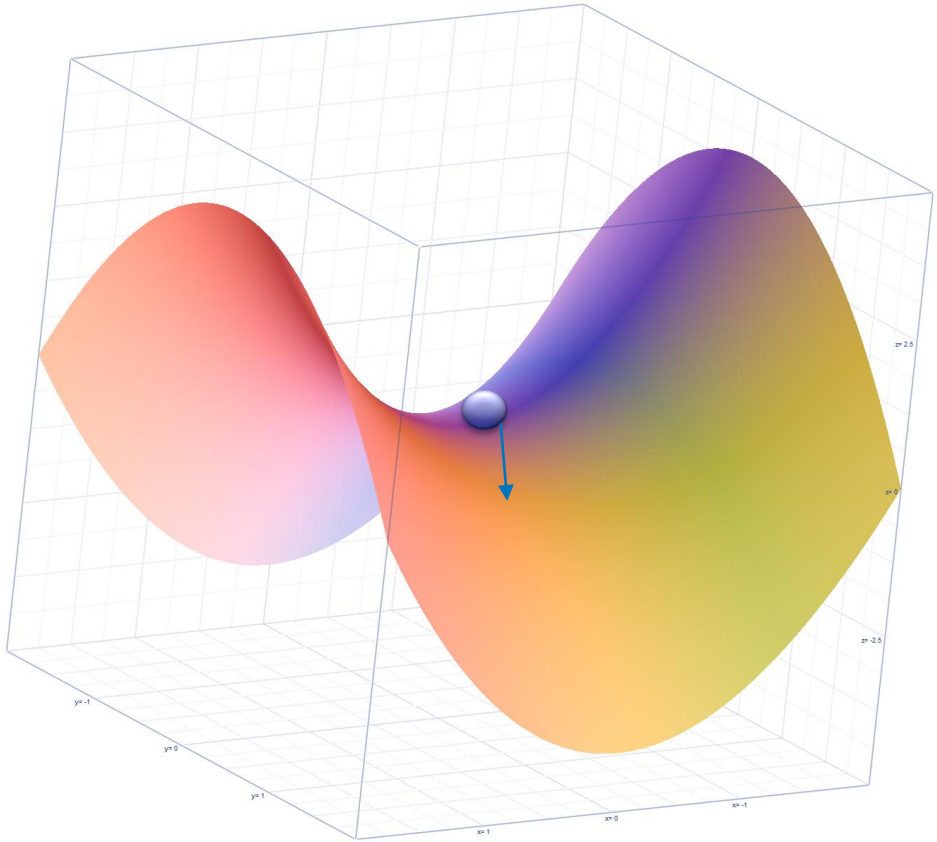
Without momentum



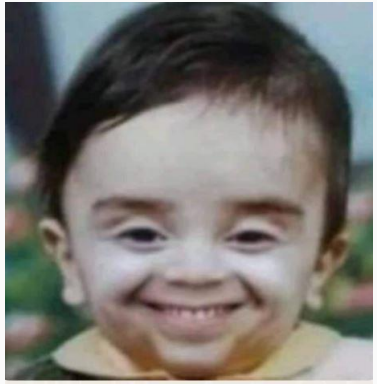
Without momentum

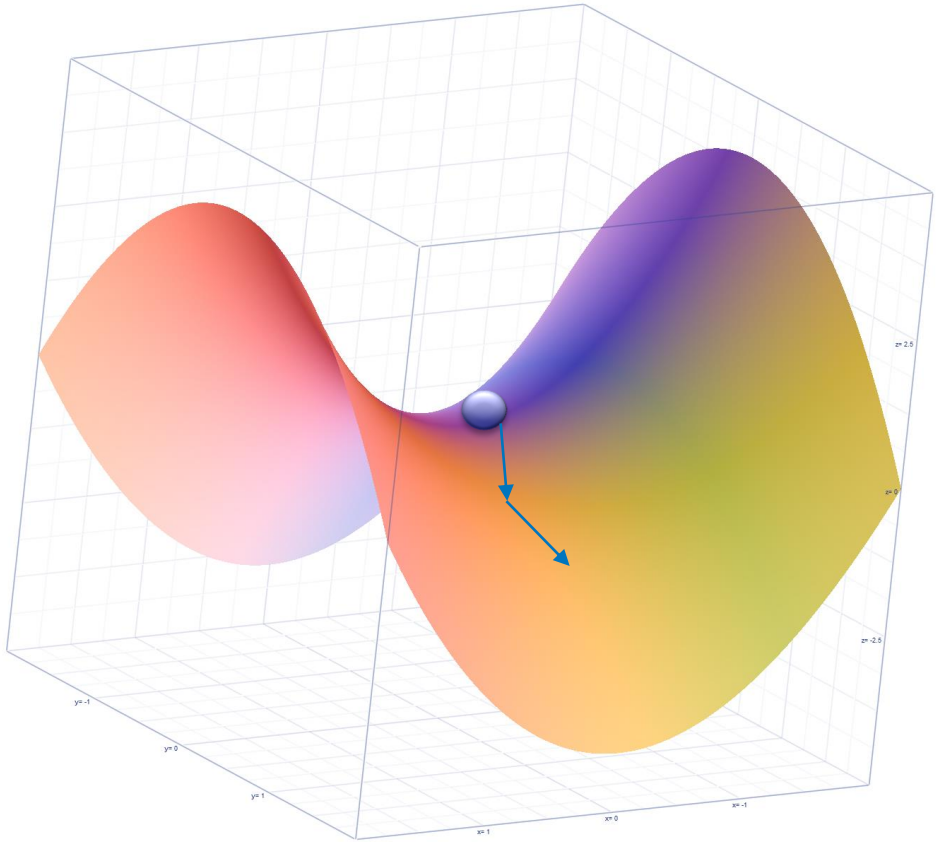

z= 2.5
z= 0
z= -2.5
y= -1
y= 0
y= 1
x= 1
x= 0
x= -1



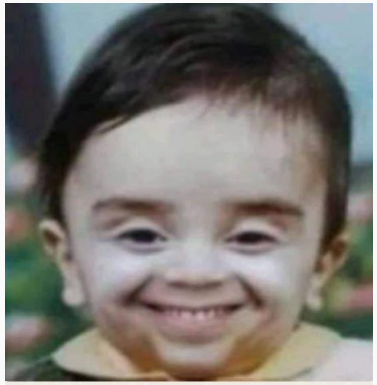


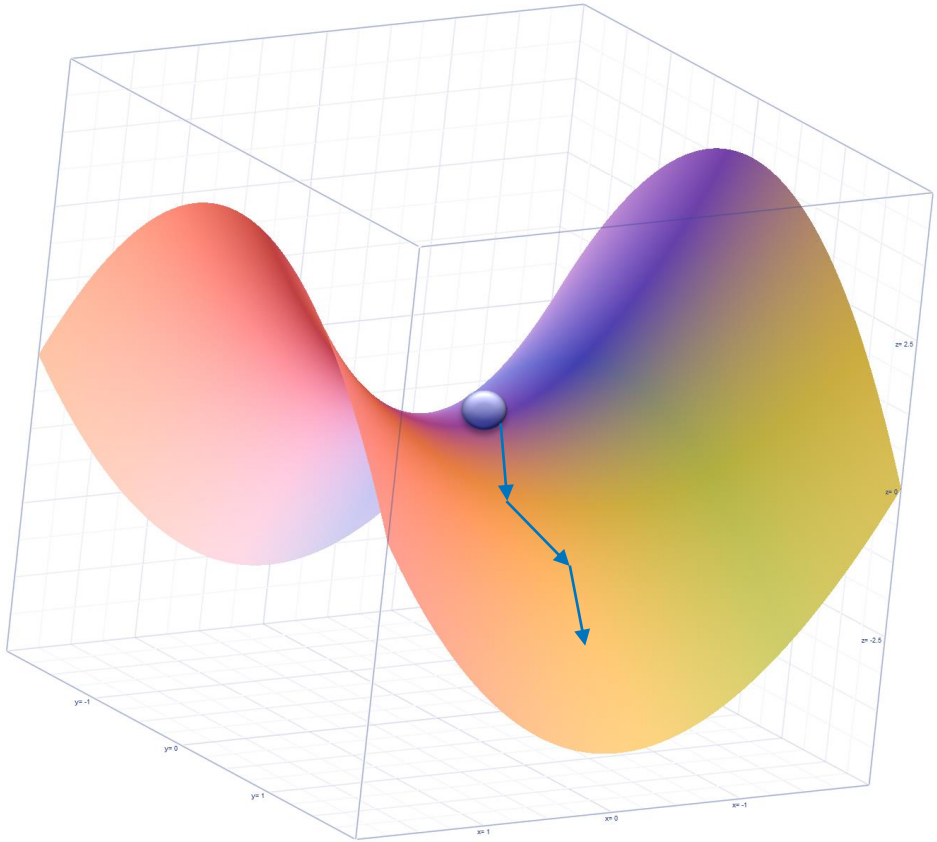
With momentum



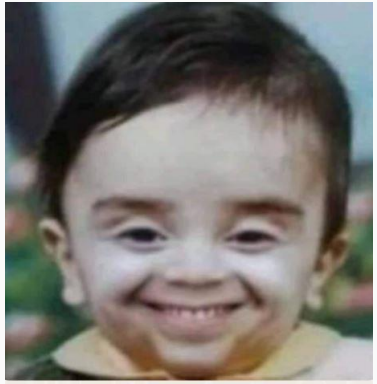


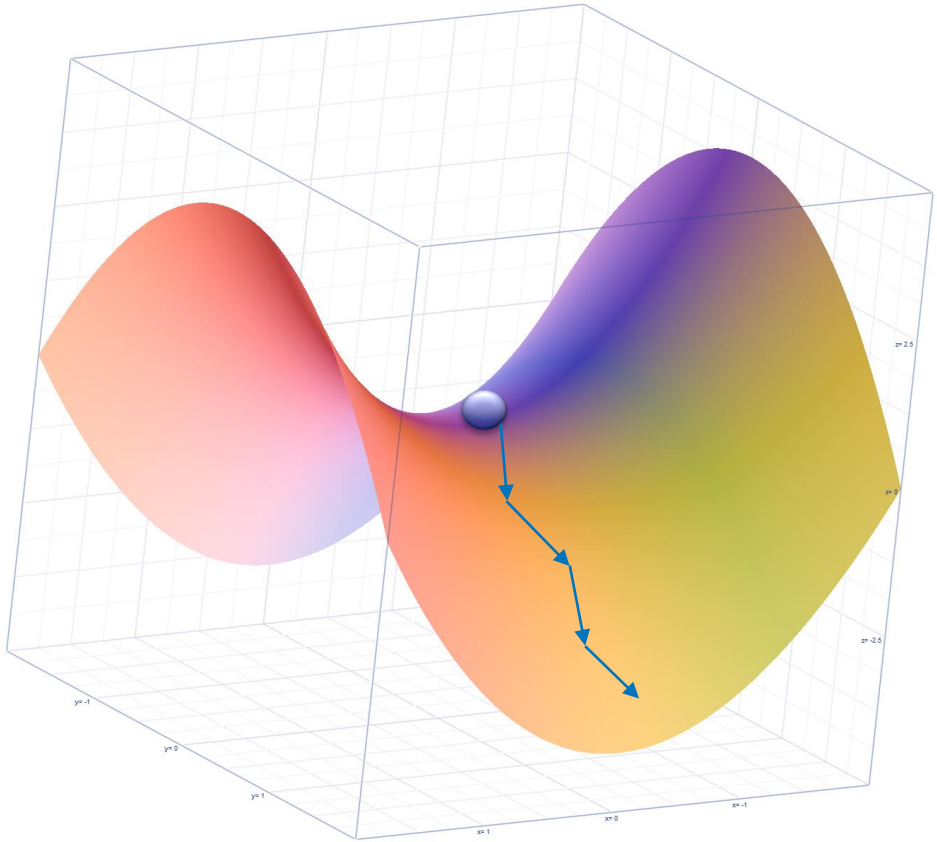
With momentum



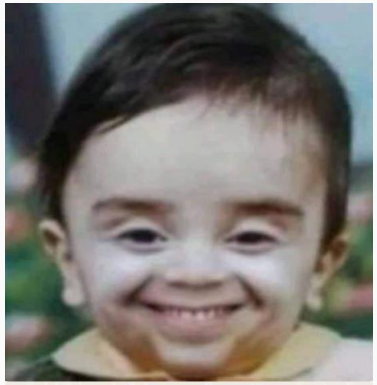


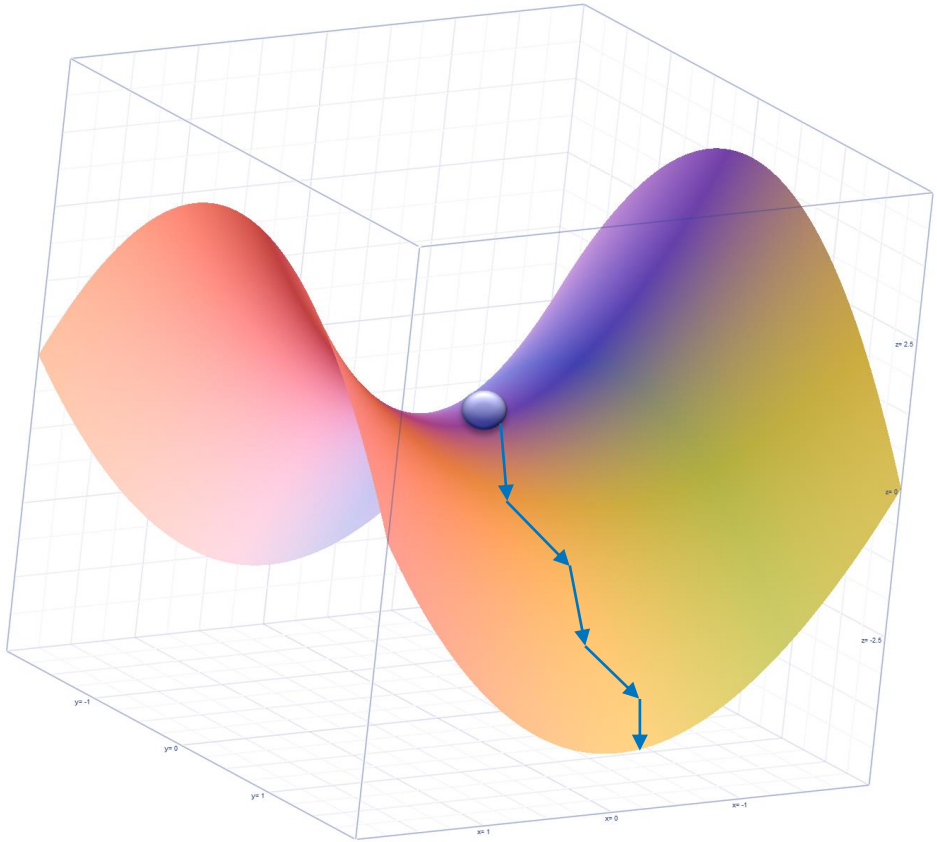
With momentum





With momentum





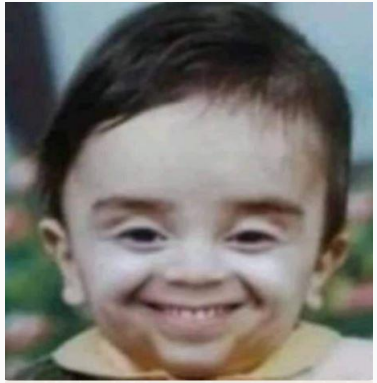
With momentum

Without momentum

$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$

With momentum

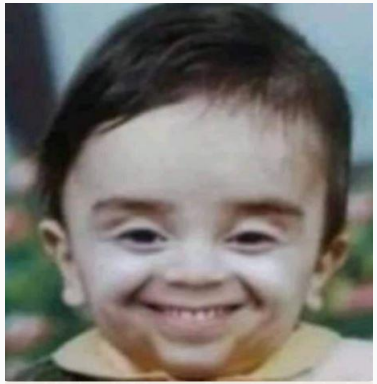
$$m = \beta m + \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$



$$w_{ij}^{(L)} = w_{ij}^{(L)} - m$$

With momentum

$$m = \beta m + \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$

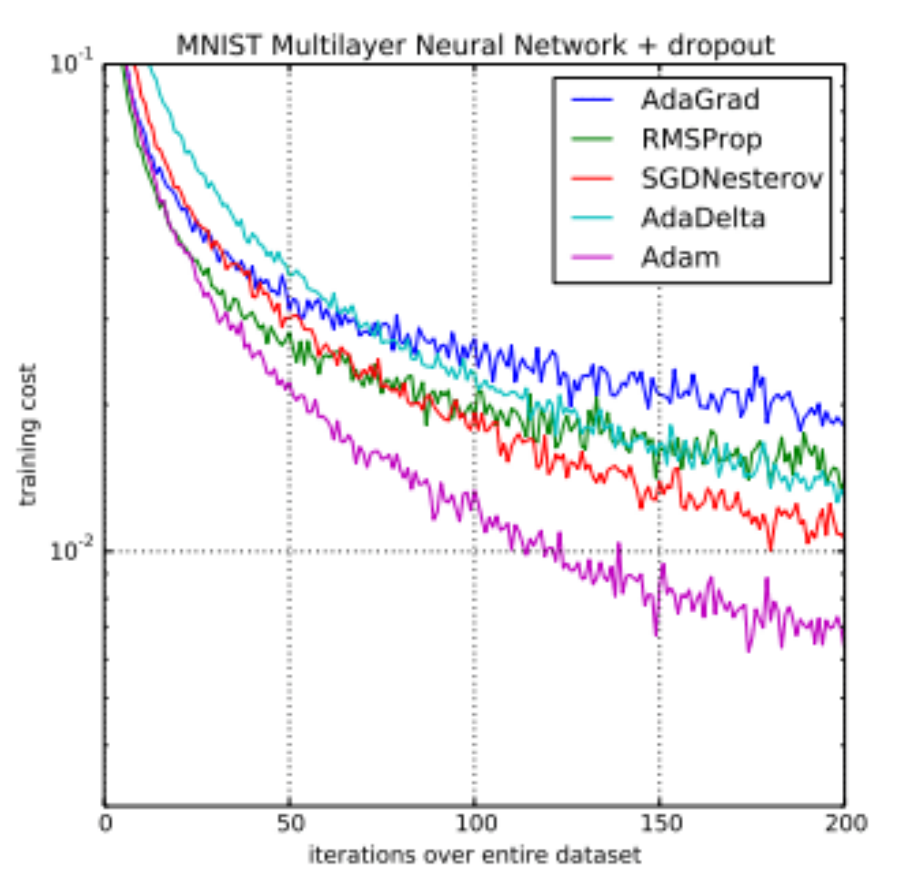


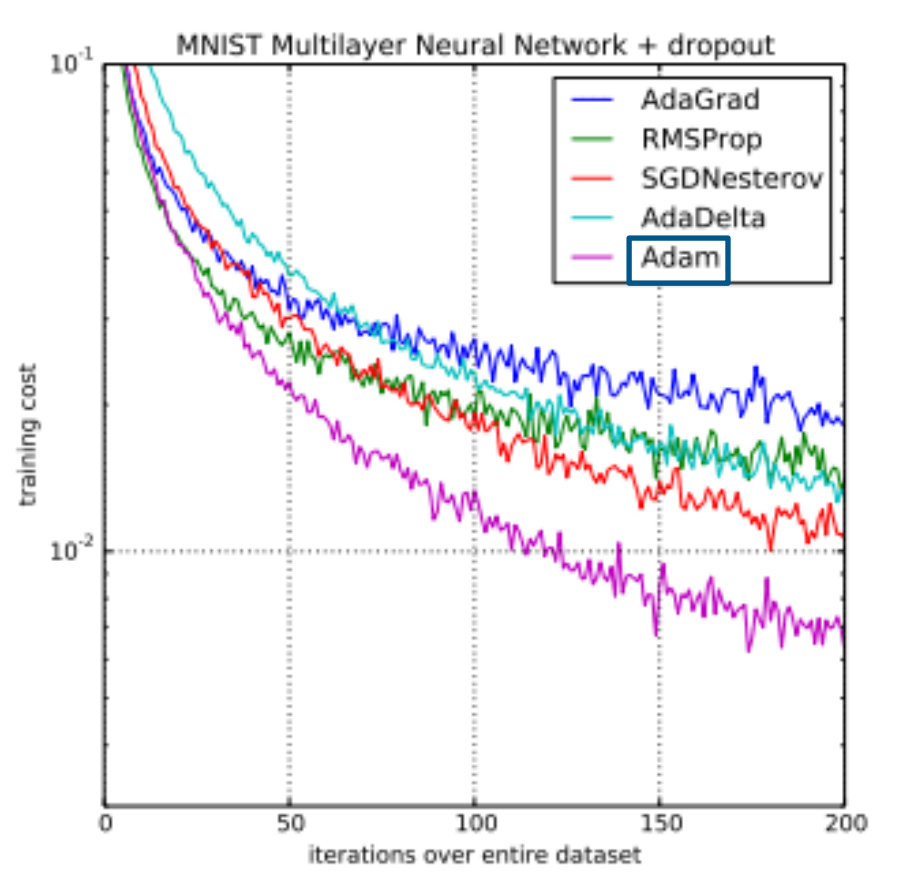
$$w_{ij}^{(L)} = w_{ij}^{(L)} - m$$

β : Friction

$\beta = 0: high\ friction$

$\beta = 1: low\ friction$

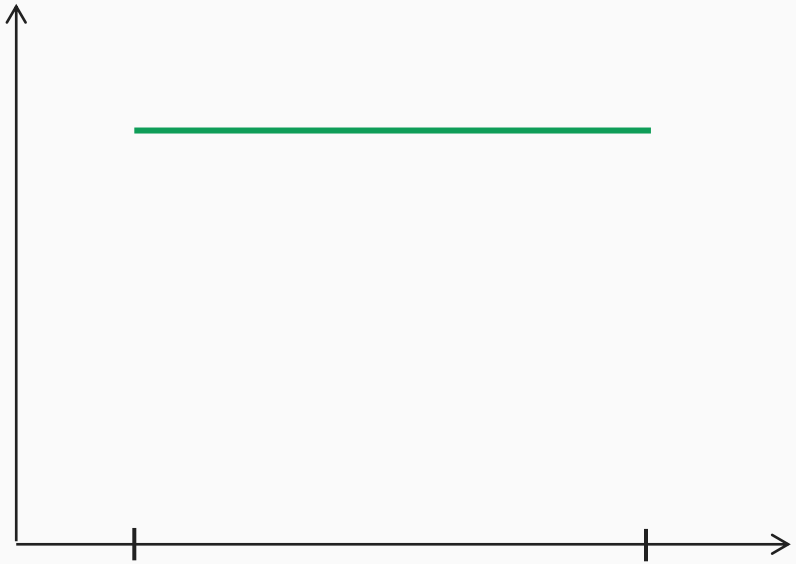

MNIST Multilayer Neural Network + dropout
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
training cost
10^{-1}
10^{-2}
0
50
100
150
200
iterations over entire dataset


MNIST Multilayer Neural Network + dropout
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
training cost
10^{-1}
10^{-2}
0
50
100
150
200
iterations over entire dataset

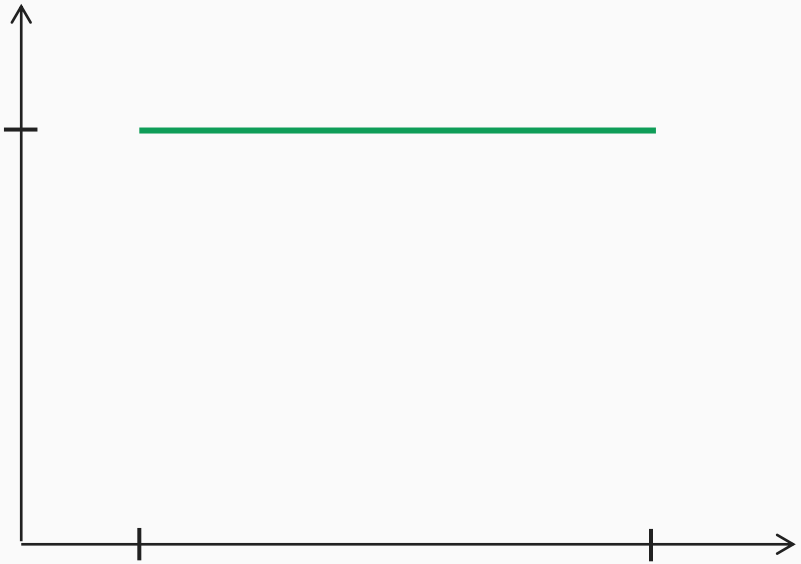
Weight initialization

Randomly using uniform distribution

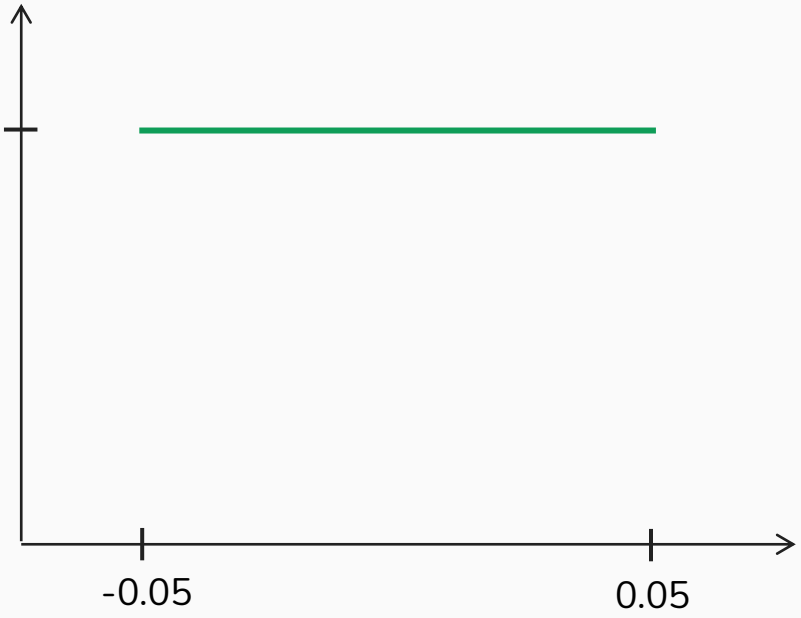
Randomly using uniform distribution



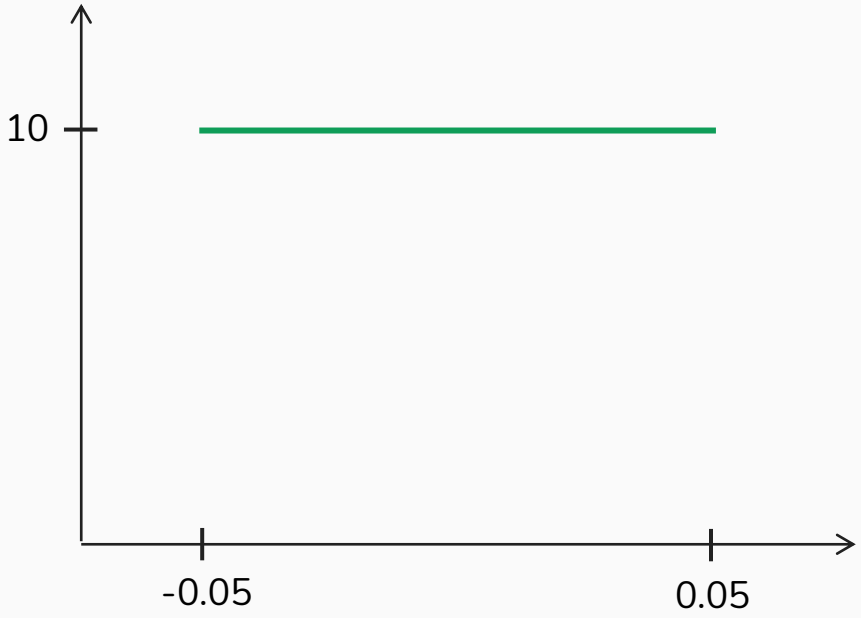
Randomly using uniform distribution



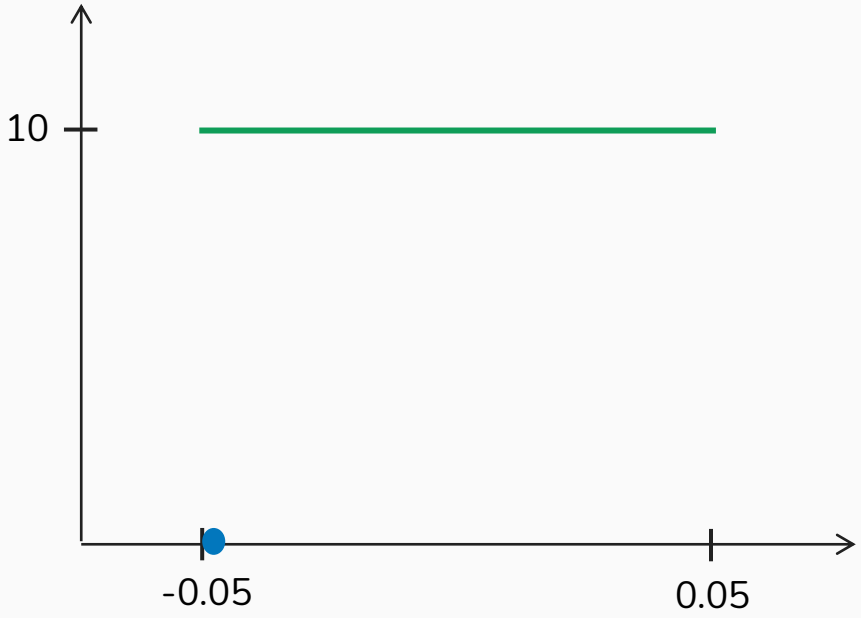
Randomly using uniform distribution



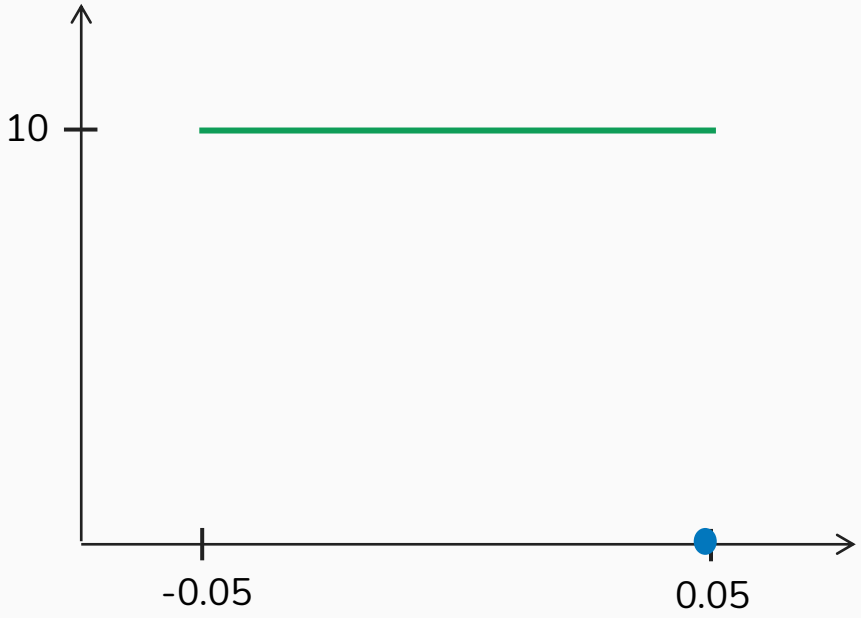
Randomly using uniform distribution



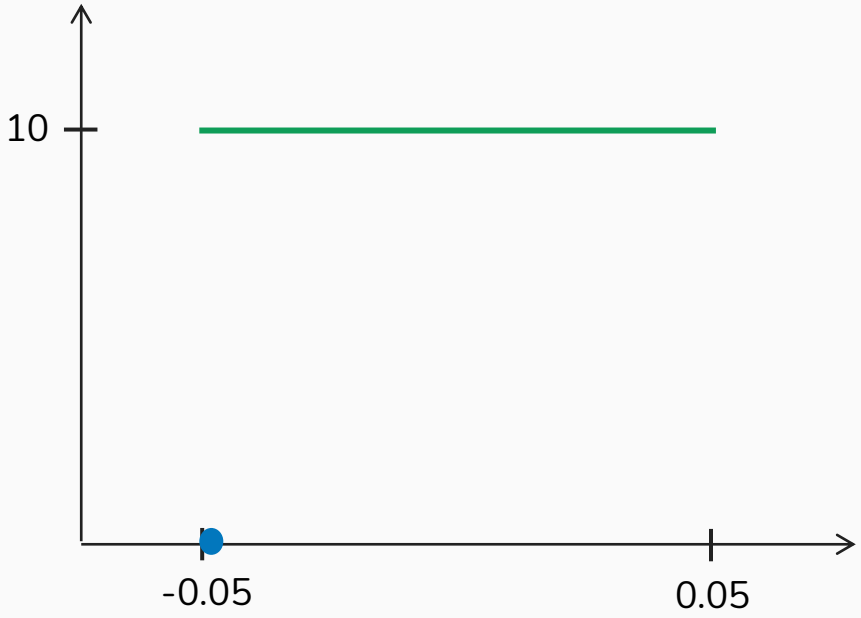
Randomly using uniform distribution



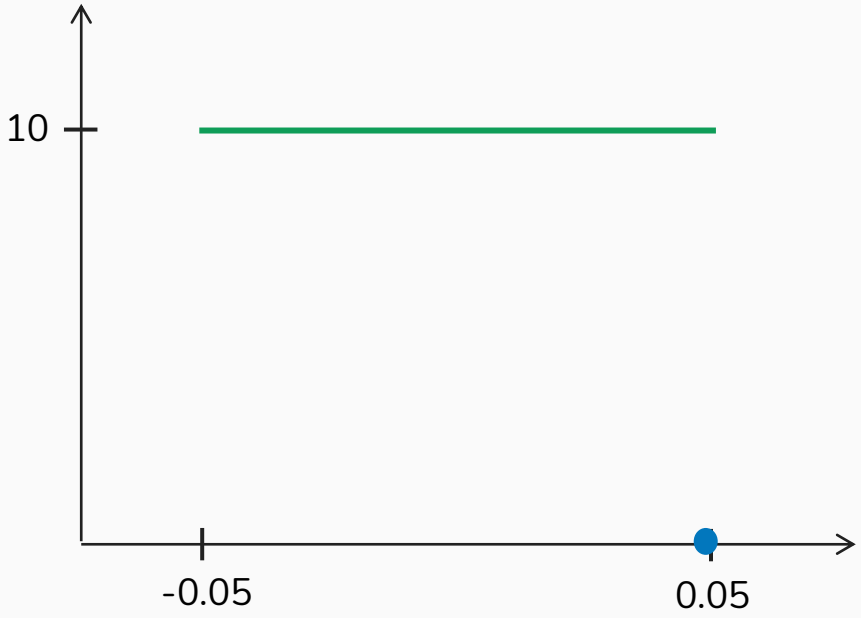
Randomly using uniform distribution



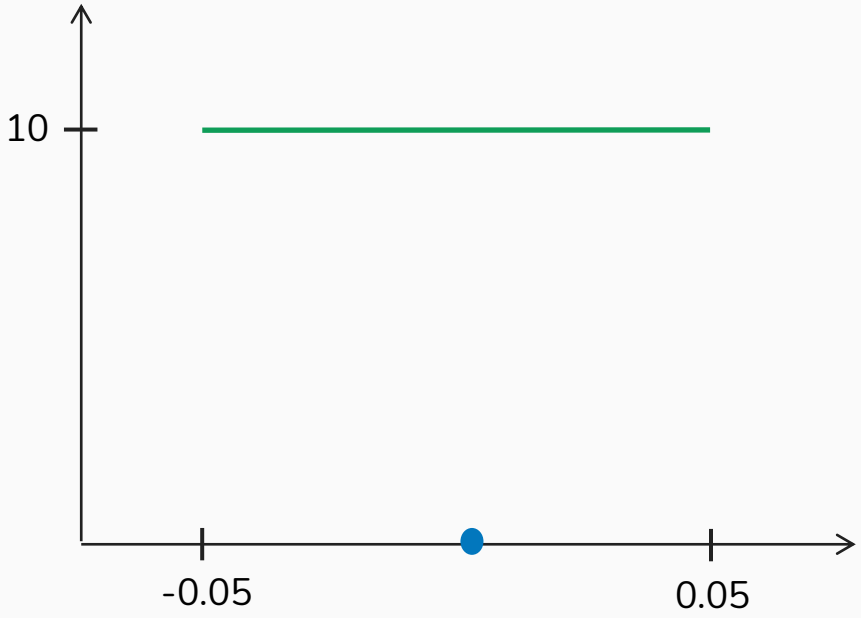
Randomly using uniform distribution



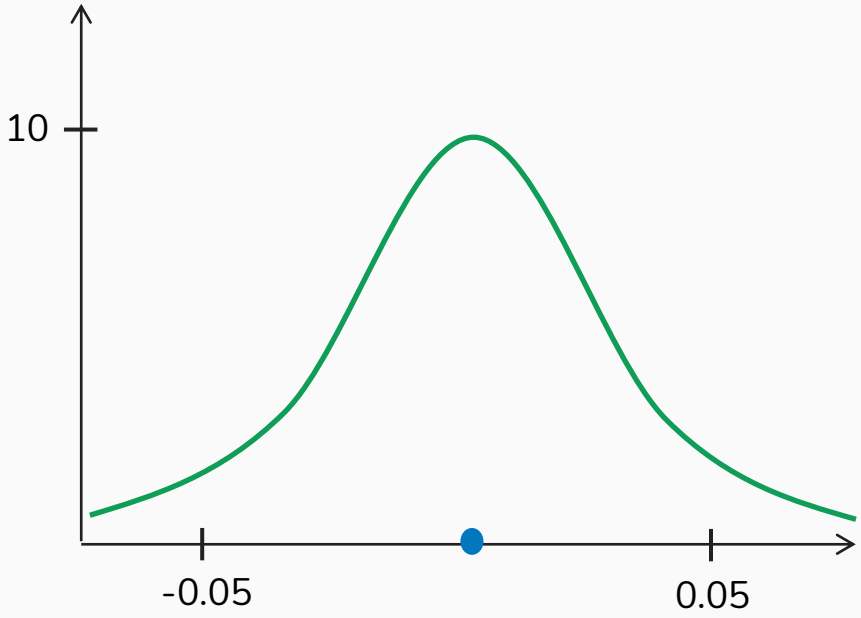
Randomly using uniform distribution



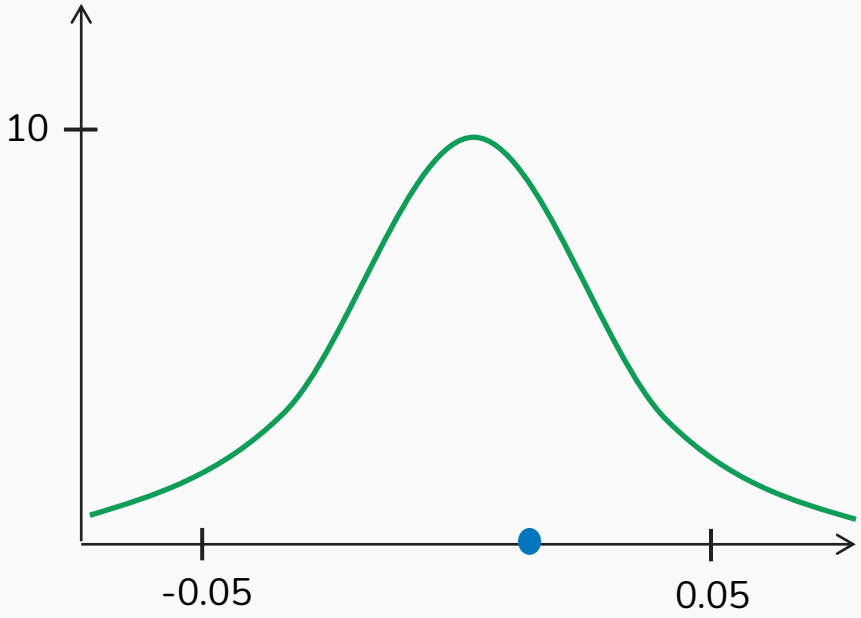
Randomly using uniform distribution



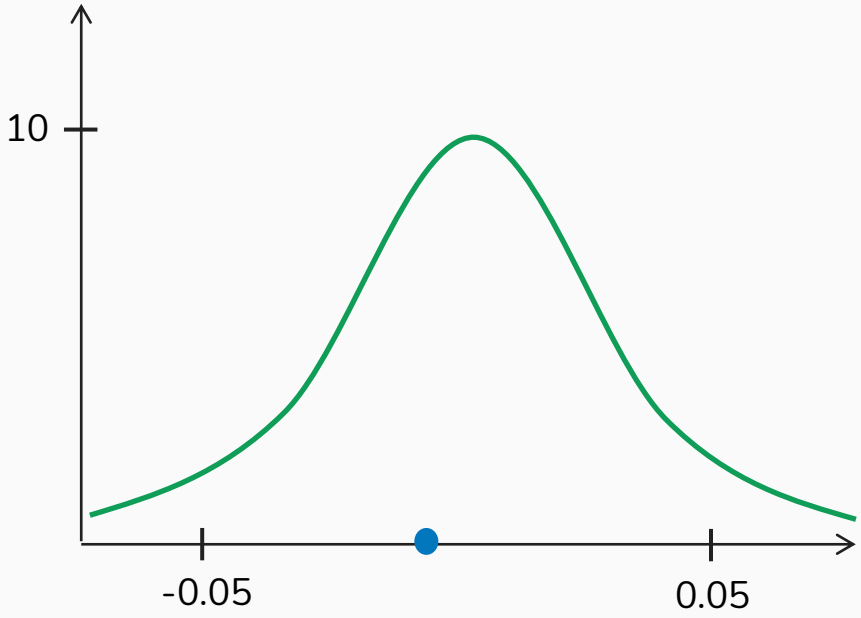
Randomly using normal distribution



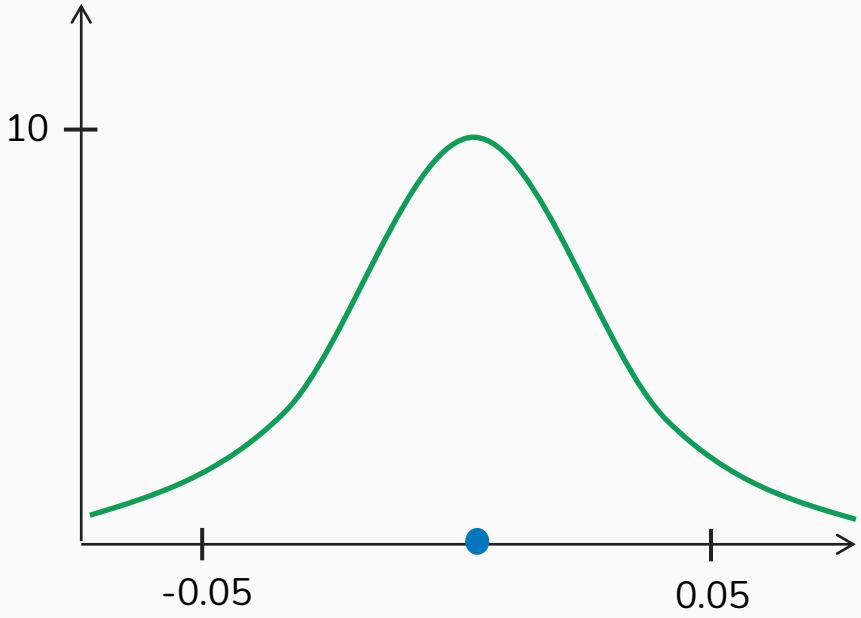
Randomly using normal distribution



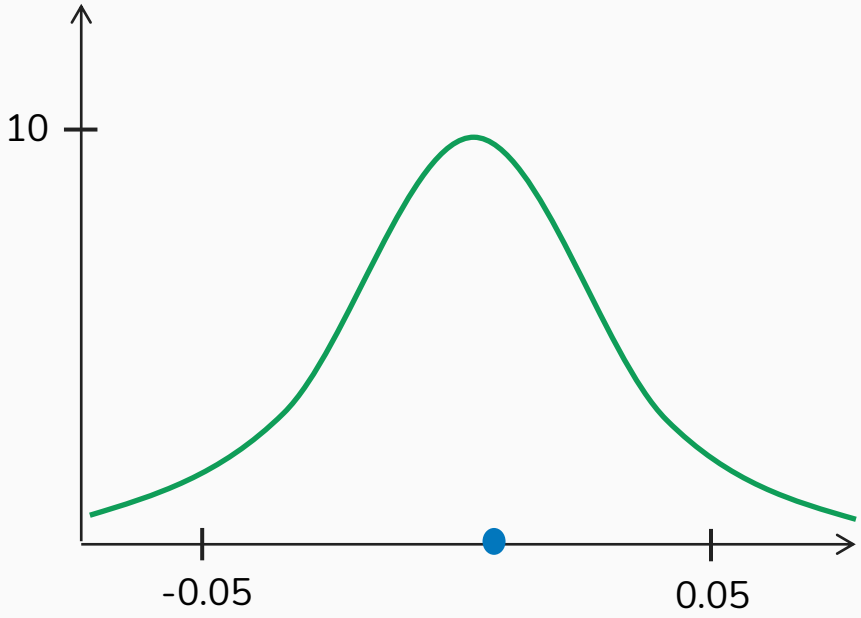
Randomly using normal distribution



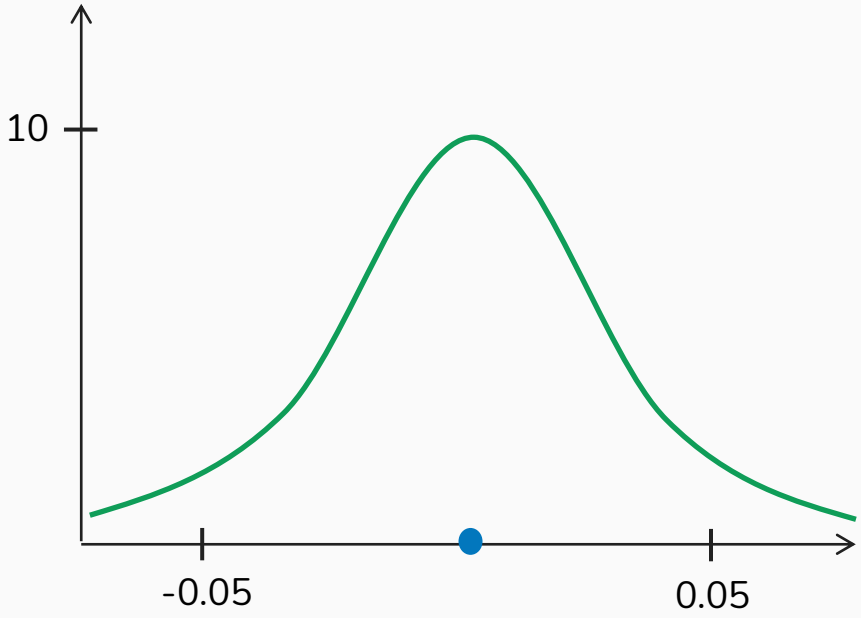
Randomly using normal distribution



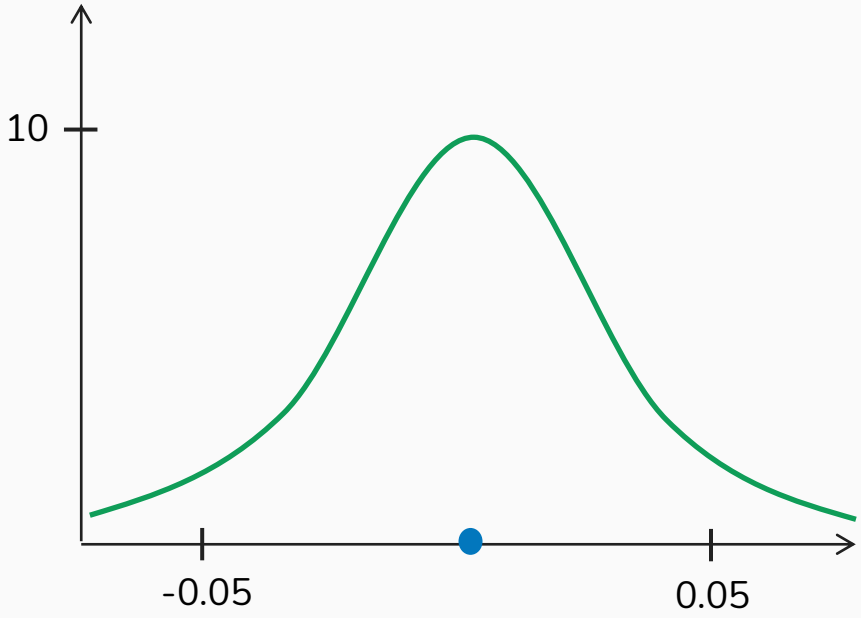
Randomly using normal distribution



Randomly using normal distribution



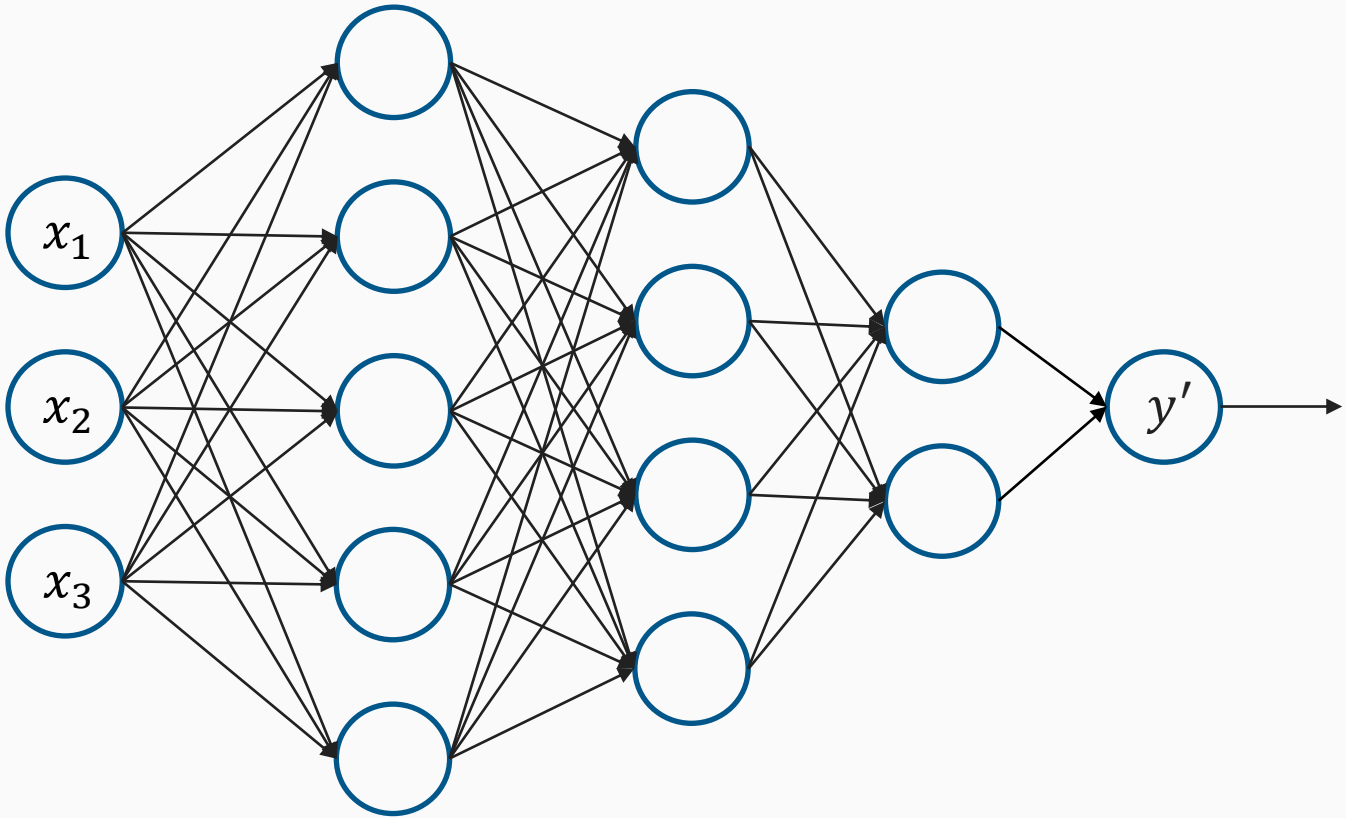
Randomly using normal distribution

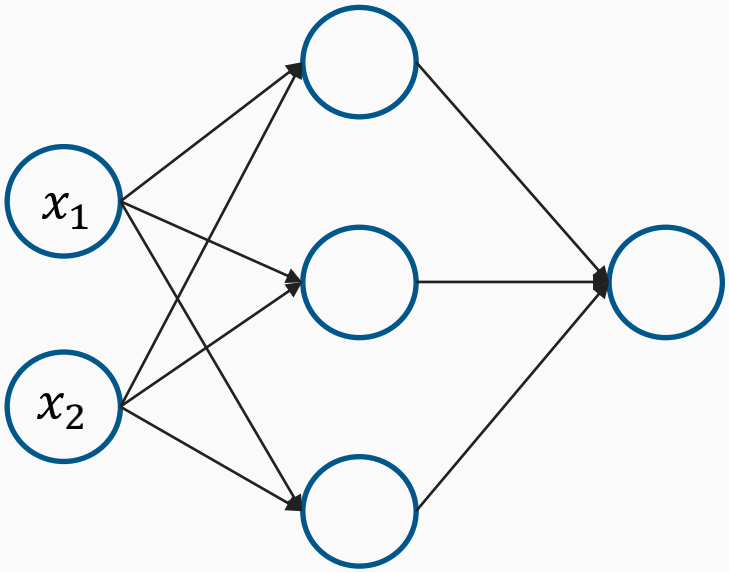


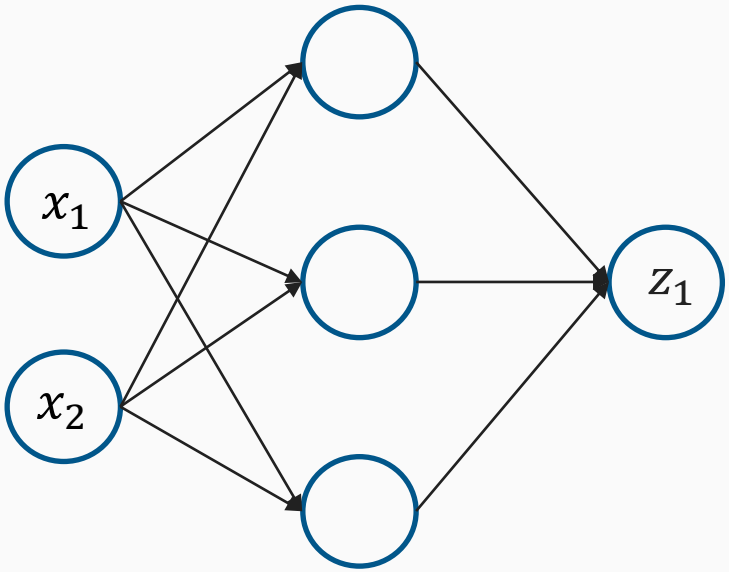
We use them rarely

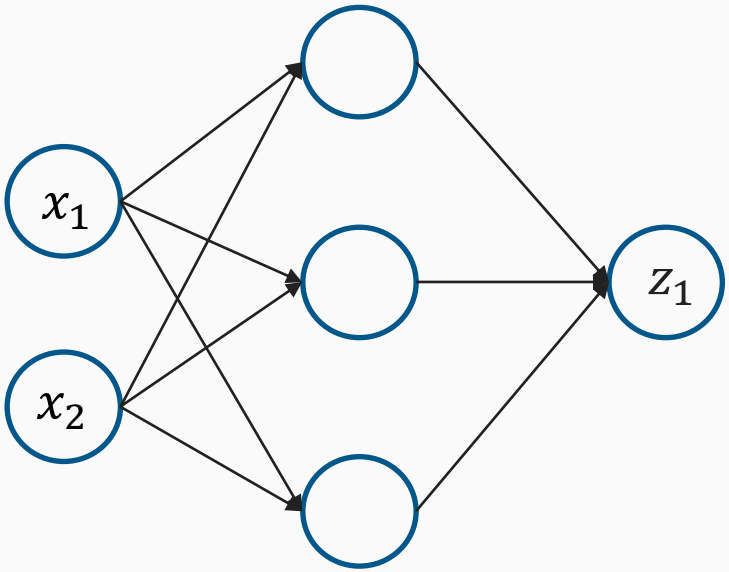
Why ?

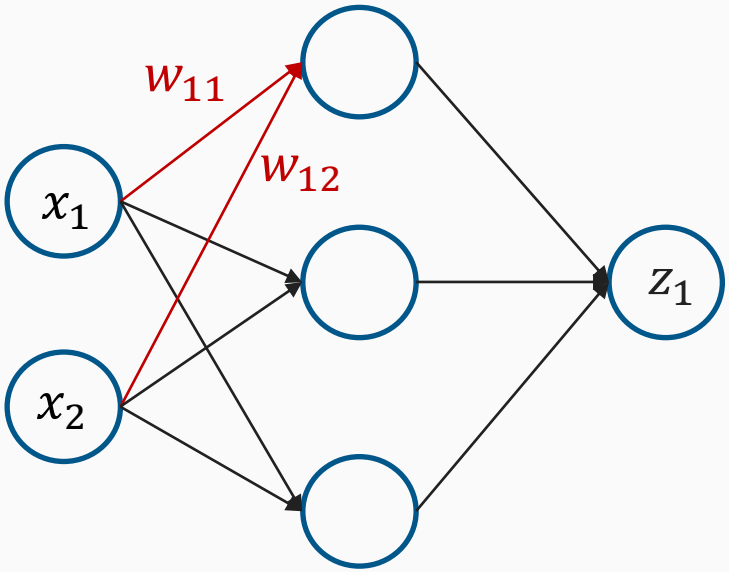
The vanishing and exploding gradients

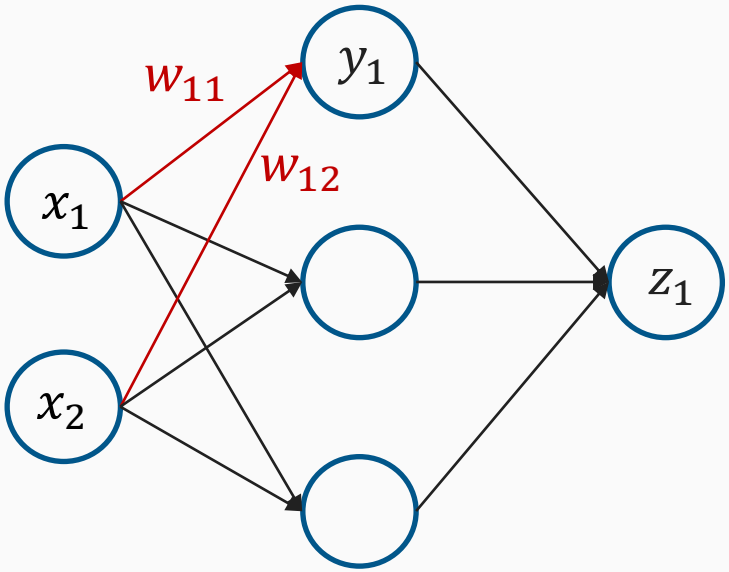

x_1
x_2
x_3
y'


x_1
x_2

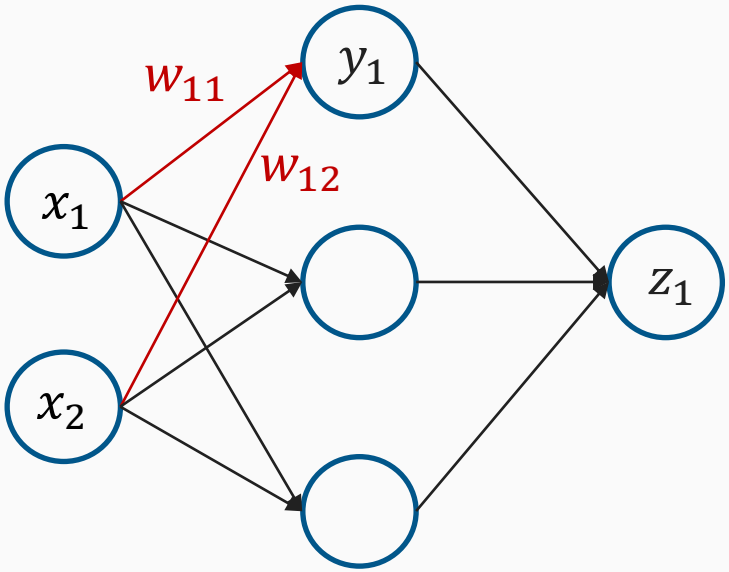

x_1
x_2
z_1

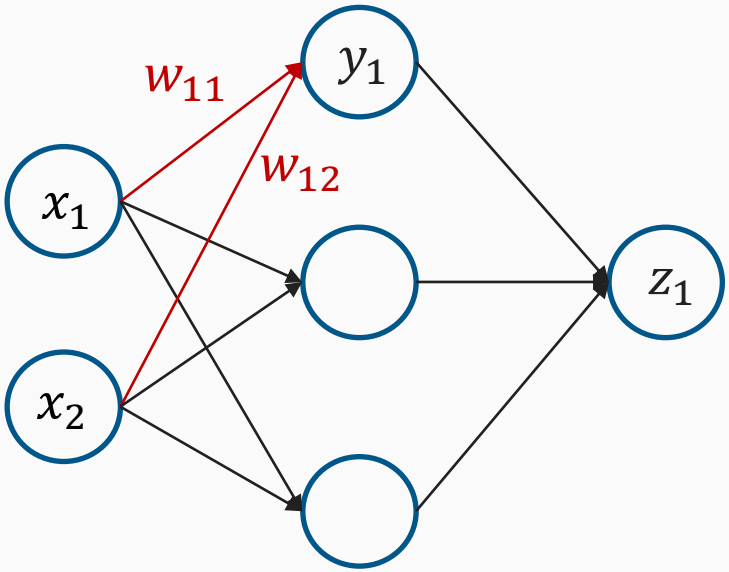

x_1
x_2
z_1


w_{11}
w_{12}
x_1
x_2
z_1


w_{11}
w_{12}
y_1
x_1
x_2
z_1

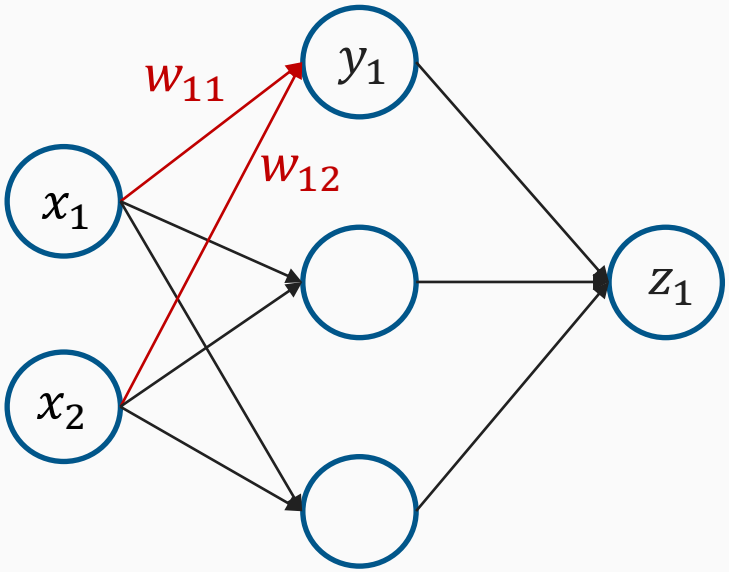
$$y_1 = x_1 w_{11} + x_2 w_{12}$$





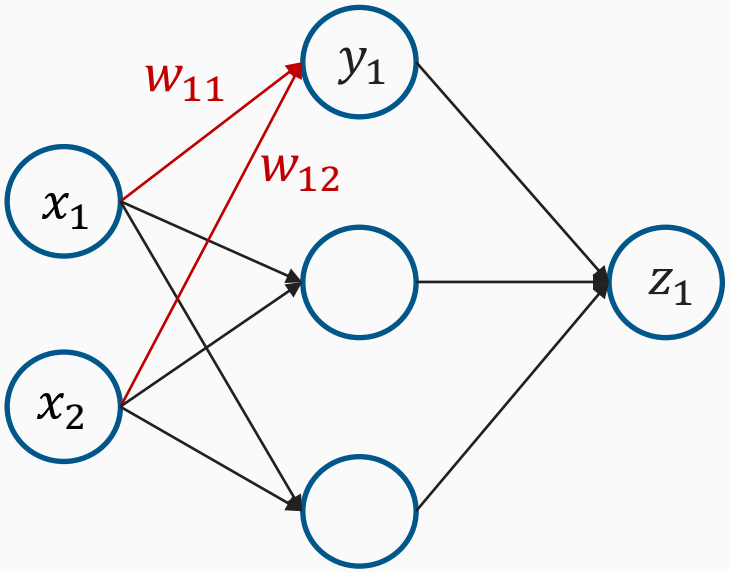
$$y_1 = x_1 w_{11} + x_2 w_{12}$$

$$Var(y_1) = Var(x_1 w_{11} + x_2 w_{12})$$



$$y_1 = x_1 w_{11} + x_2 w_{12}$$

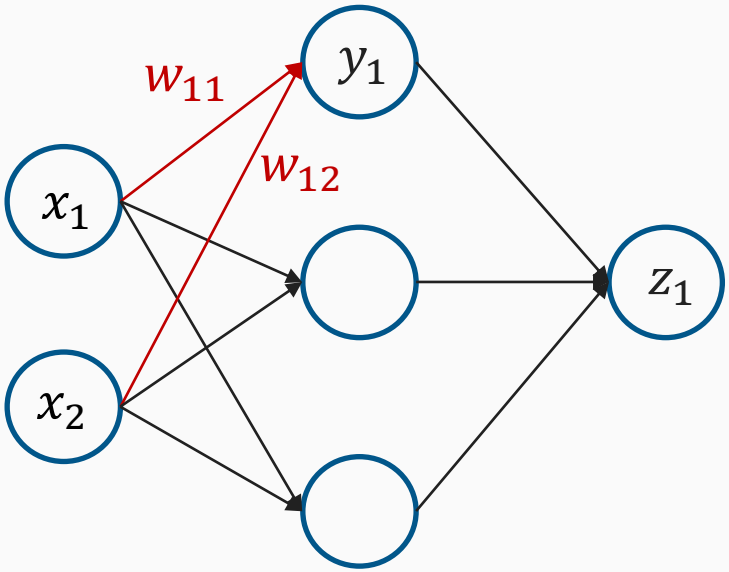
$$Var(y_1) = Var(x_1 w_{11} + x_2 w_{12})$$



$$y_1 = x_1 w_{11} + x_2 w_{12}$$

$$Var(y_1) = Var(x_1 w_{11} + x_2 w_{12})$$

$$Var(y_1) = Var(x_1 w_{11}) + Var(x_2 w_{12}) + 2Cov[x_1 w_{11}, x_2 w_{12}]$$

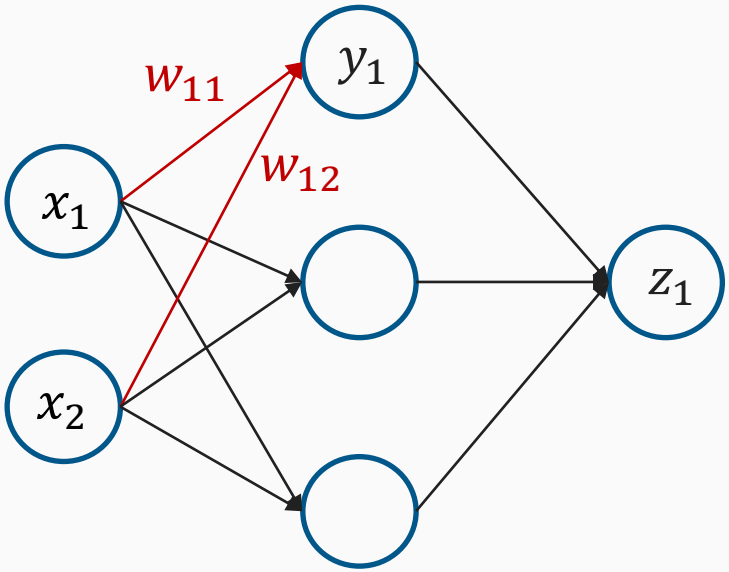


$$y_1 = x_1 w_{11} + x_2 w_{12}$$

$$Var(y_1) = Var(x_1 w_{11} + x_2 w_{12})$$

$$Var(y_1) = Var(x_1 w_{11}) + Var(x_2 w_{12}) + 2Cov[x_1 w_{11}, x_2 w_{12}]$$

$$Var(y_1) = Var(x_1 w_{11}) + Var(x_2 w_{12})$$



$$y_1 = x_1 w_{11} + x_2 w_{12}$$

$$Var(y_1) = Var(x_1 w_{11} + x_2 w_{12})$$

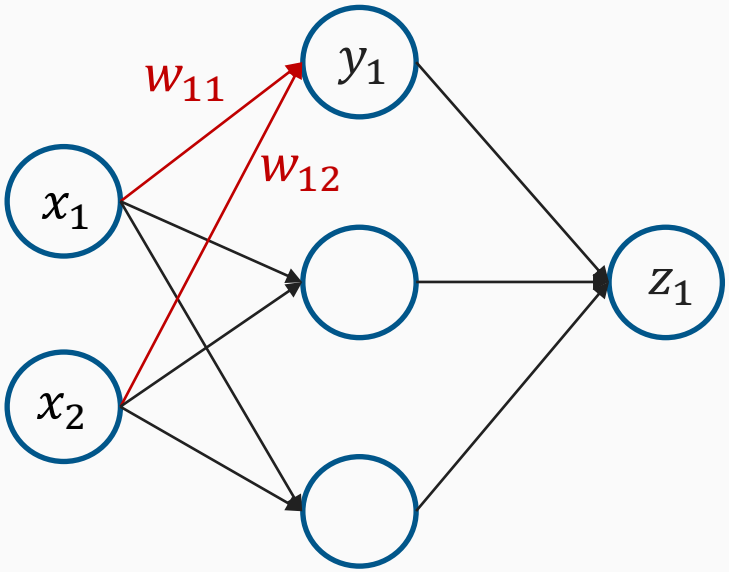
$$Var(y_1) = Var(x_1 w_{11}) + Var(x_2 w_{12}) + 2Cov[x_1 w_{11}, x_2 w_{12}]$$

$$Var(y_1) = Var(x_1 w_{11}) + Var(x_2 w_{12})$$

$$Var(y_1) = E[x_1{}^2]Var(w_{11}) + E[w_{11}{}^2]Var(x_1) + Var(x_1)Var(w_{11}) + E[x_2{}^2]Var(w_{12}) + E[w_{12}{}^2]Var(x_2) + Var(x_2)Var(w_{12})$$

Suppose x and w have zero mean

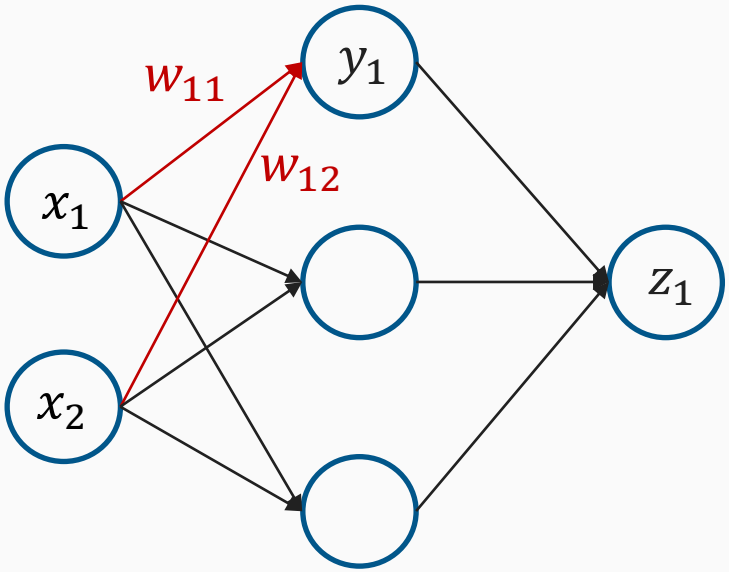
$$E[x] = E[w] = 0$$



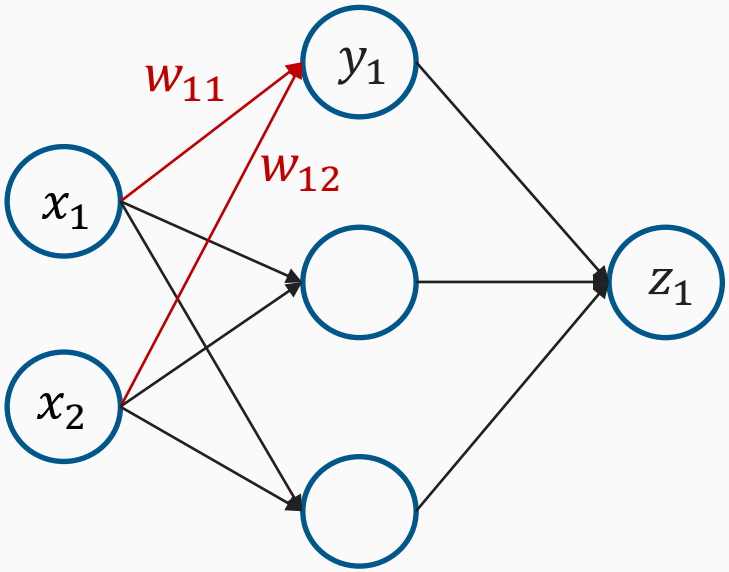
$$Var(y_1)$$
$$= E\left[{x_1}^2\right]Var(w_{11}) + E\left[{w_{11}}^2\right]Var(x_1) + Var(x_1)Var(w_{11})$$
$$+ E\left[{x_2}^2\right]Var(w_{12}) + E\left[{w_{12}}^2\right]Var(x_2) + Var(x_2)Var(w_{12})$$

Suppose x and w have zero mean

$$E[x] = E[w] = 0$$

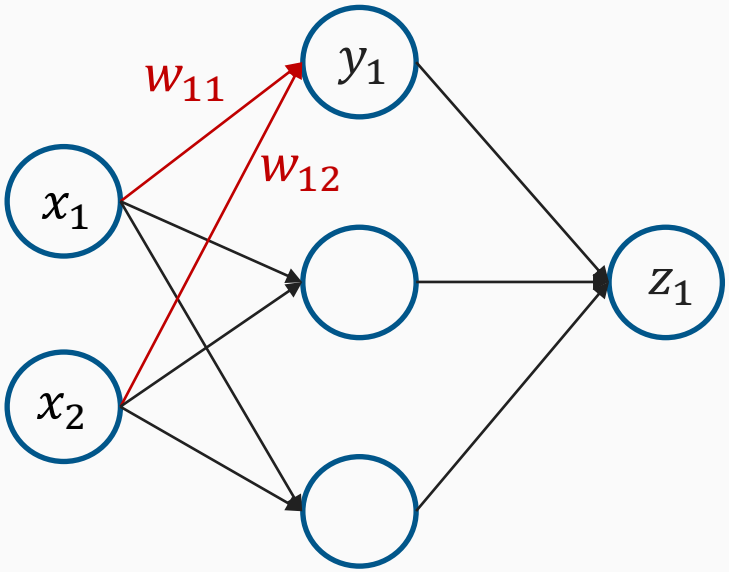


$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$



Suppose x and w have zero mean

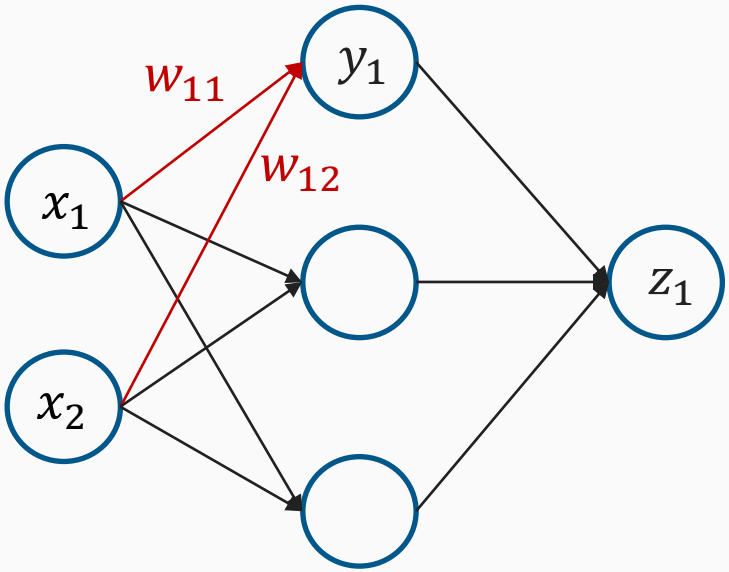
$$E[x] = E[w] = 0$$



Suppose x and w have zero mean

$$E[x] = E[w] = 0$$

$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$

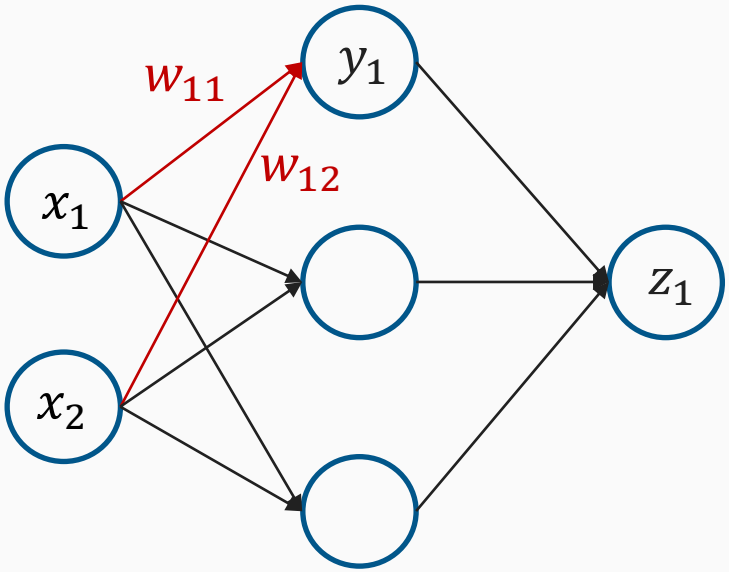


Suppose x and w have zero mean

$$E[x] = E[w] = 0$$

$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$

$$Var(y_1) = \sum_{i=1}^{2} Var(x_\mathrm{i})Var(w_{1i})$$



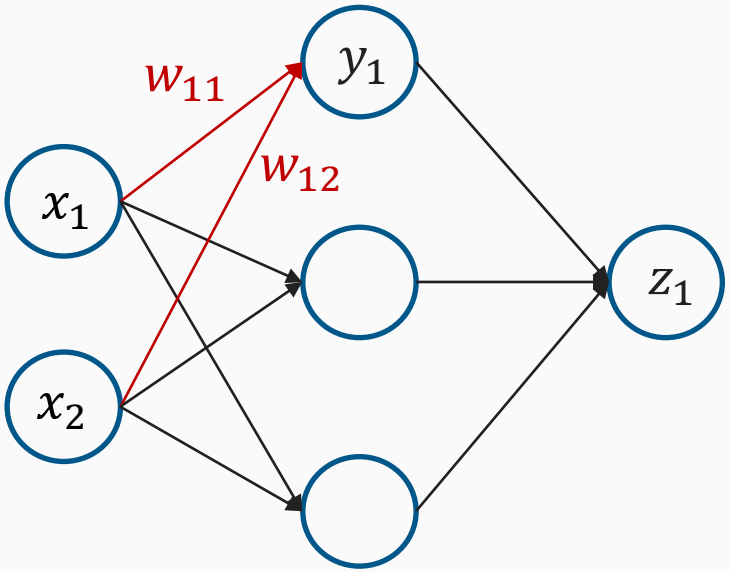
Suppose x and w have zero mean

$$E[x] = E[w] = 0$$

$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$

$$Var(y_1) = \sum_{i=1}^{2} Var(x_\mathrm{i})Var(w_{1i})$$

Since iid $\longrightarrow$



Suppose x and w have zero mean

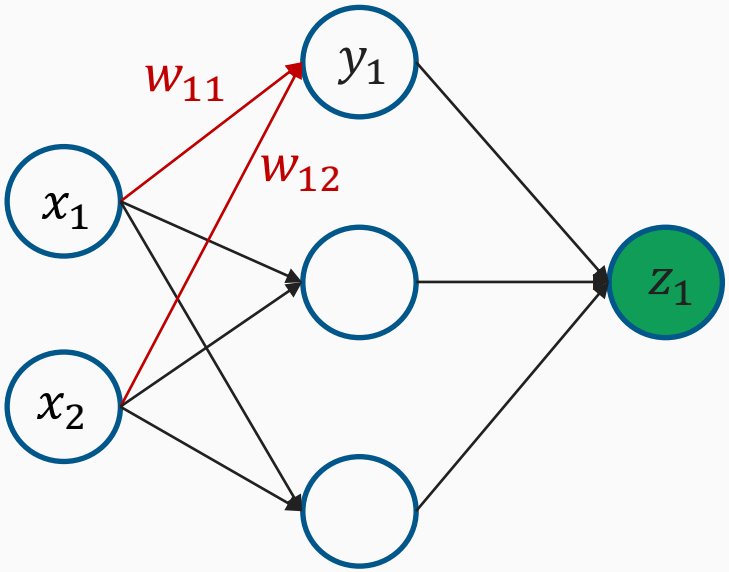
$$E[x] = E[w] = 0$$

$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$

$$Var(y_1) = \sum_{i=1}^{2} Var(x_\mathrm{i})Var(w_{1i})$$

Since iid $\longrightarrow$

$$Var(y_1) = 2Var(x_\mathrm{i})Var(w_{1i})$$



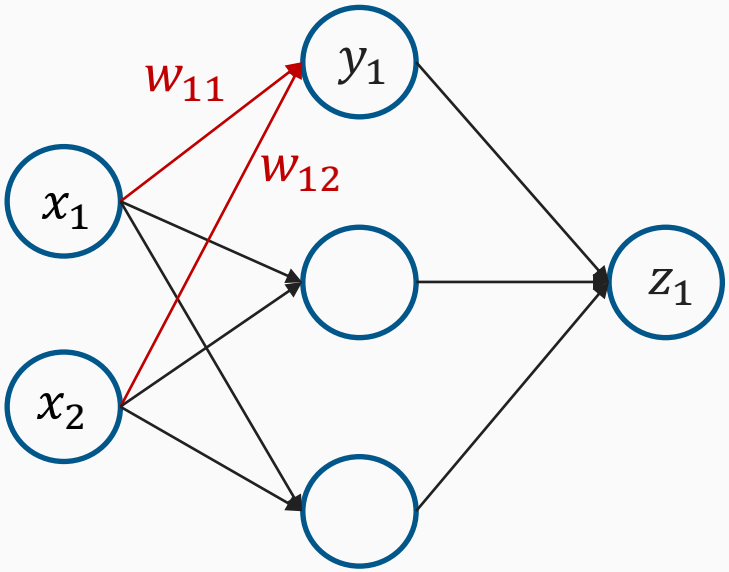
Suppose x and w have zero mean

$$E[x] = E[w] = 0$$

$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$

$$Var(y_1) = \sum_{i=1}^{2} Var(x_\mathrm{i})Var(w_{1i})$$

Since iid $\longrightarrow$ $$Var(y_1) = 2Var(x_\mathrm{i})Var(w_{1i})$$



Suppose x and w have zero mean

$$E[x] = E[w] = 0$$

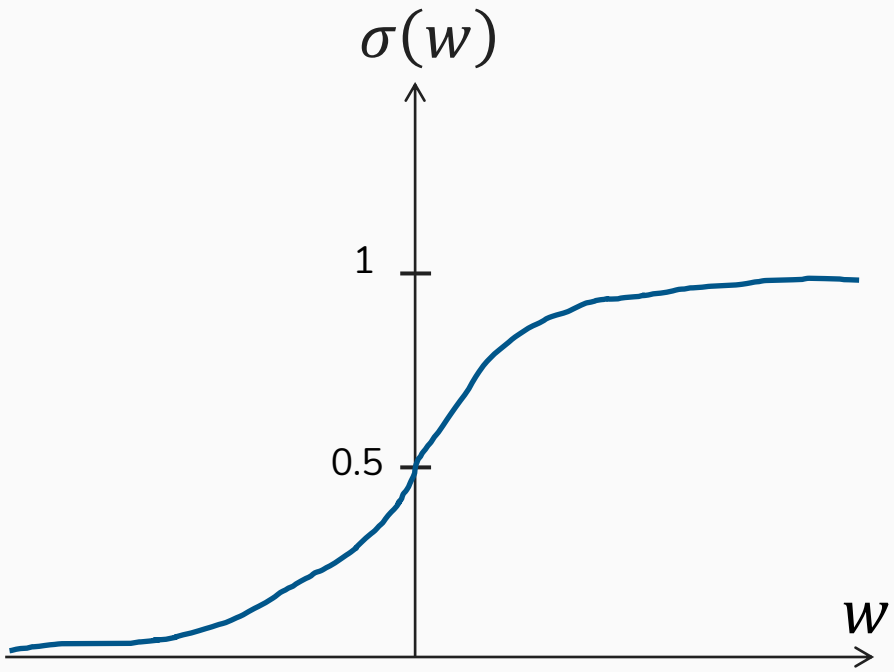
$$Var(y_1) = Var(x_1)Var(w_{11}) + Var(x_2)Var(w_{12})$$

$$Var(y_1) = \sum_{i=1}^{2} Var(x_\mathrm{i})Var(w_{1i})$$

Since iid $\longrightarrow$

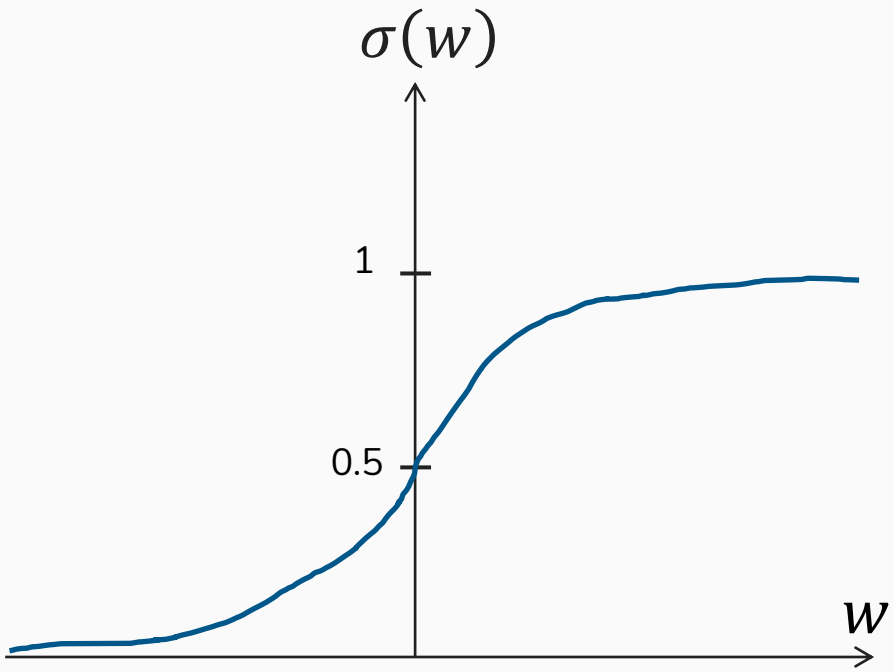
$$Var(y_1) = 2Var(x_\mathrm{i})Var(w_{1i})$$

For example : if we have a sigmoid activation



$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial J}{\partial w_{ij}^{(L)}}$$

For example : if we have a sigmoid activation



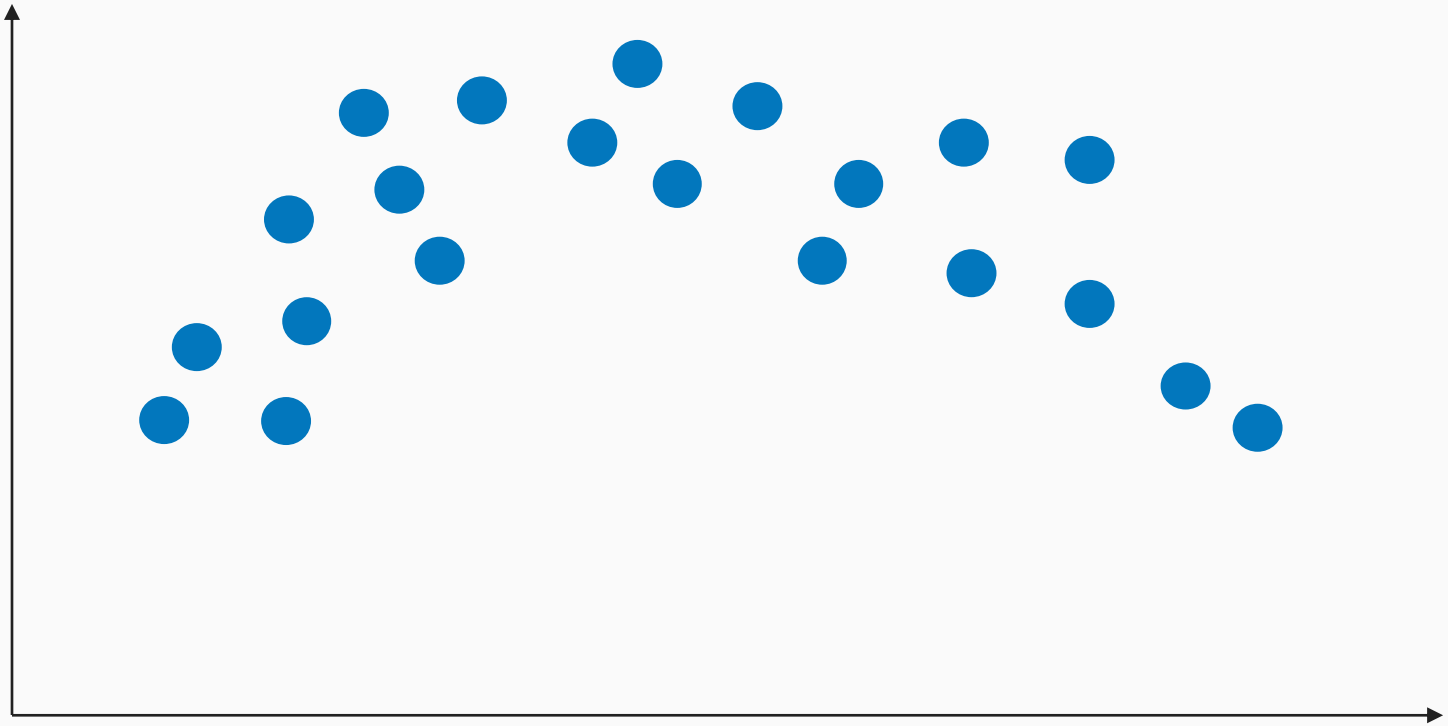
$$w_{ij}^{(L)} = w_{ij}^{(L)}$$

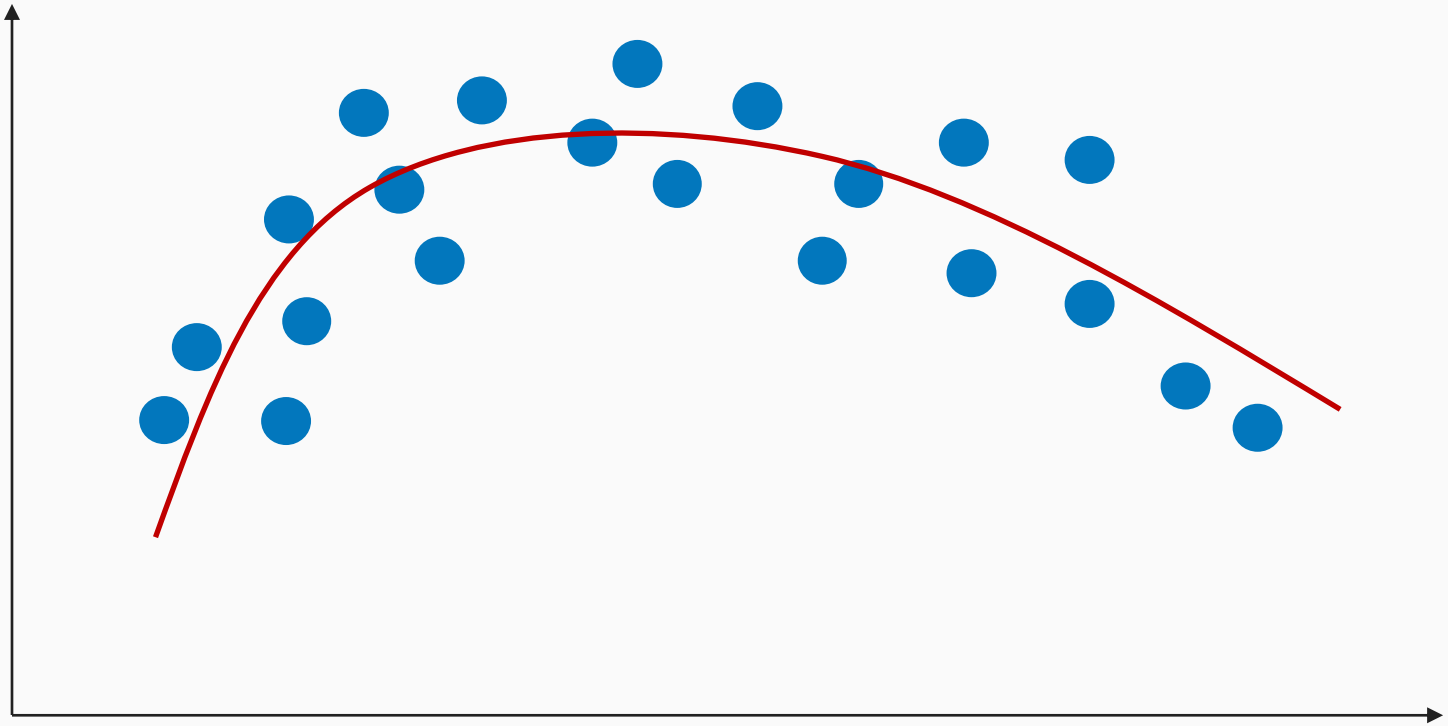
leCun initizalization

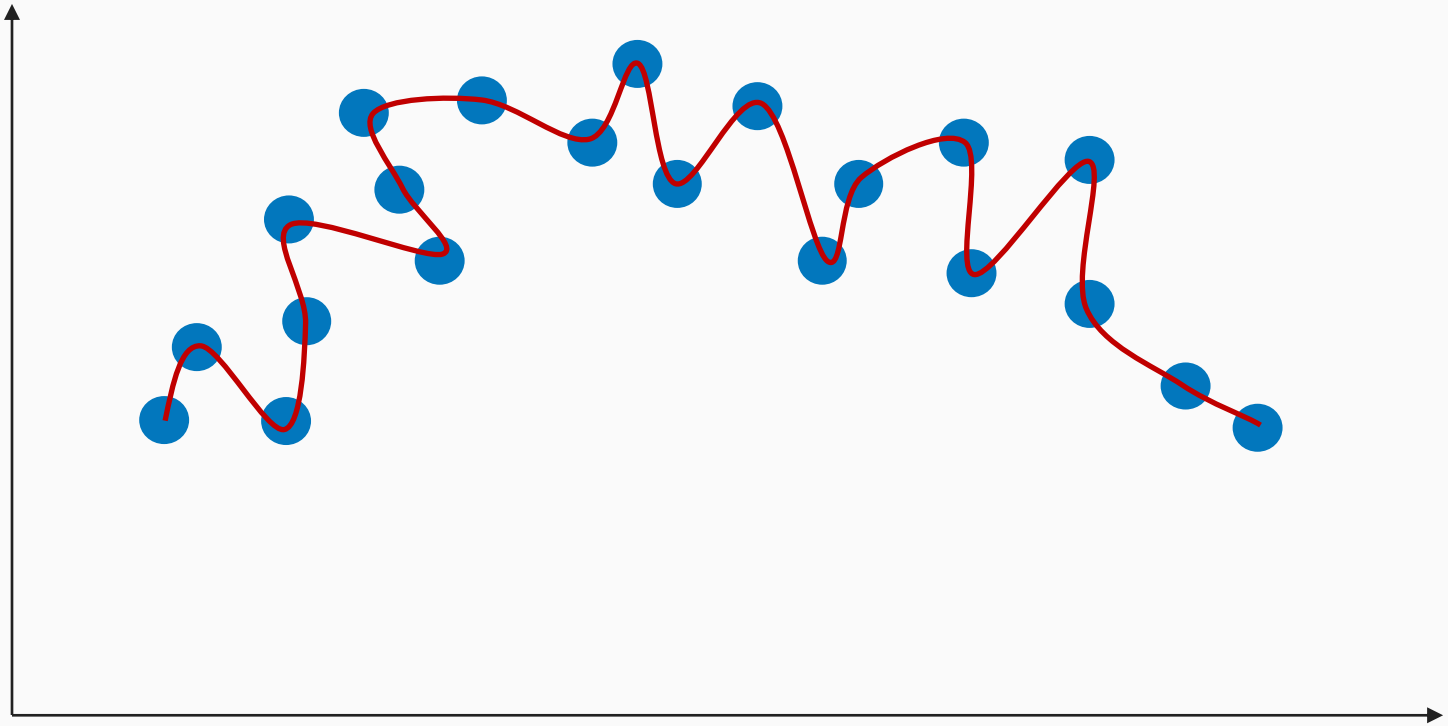
Xavier initialization

He initialization

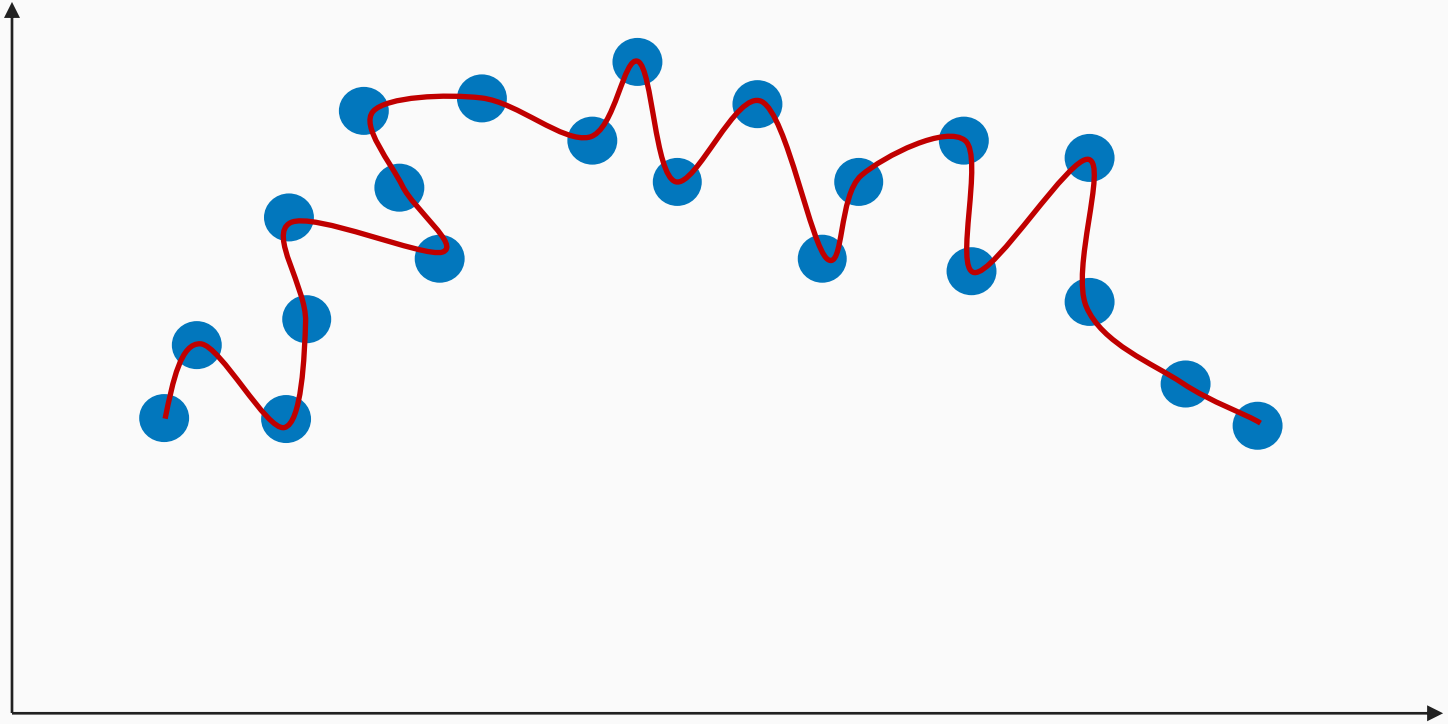
Solutions to overfitting, vanishing and exploding gradients







overfitting



over-fitting occurs when a model begins to "memorize" training data rather than "learning" to generalize

solutions

Dropout

Disconnect temporarily some of the neurons on the previous layer

Dropout

Disconnect temporarily some of the neurons on the previous layer

In training

Norm penalties

Instead of minimizing

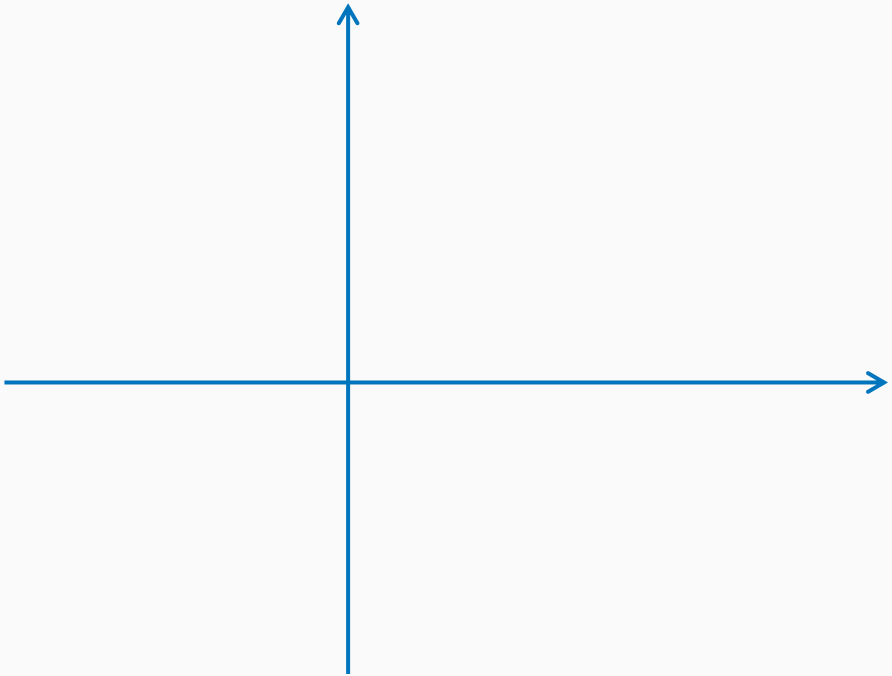
$$J$$

We minimize

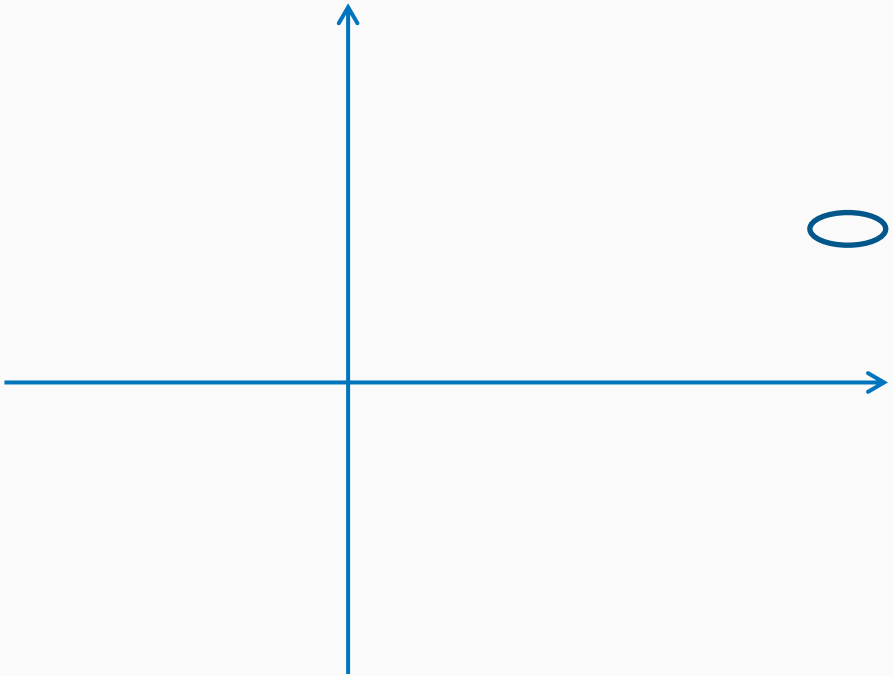
$$J + \Omega(w)$$

If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization

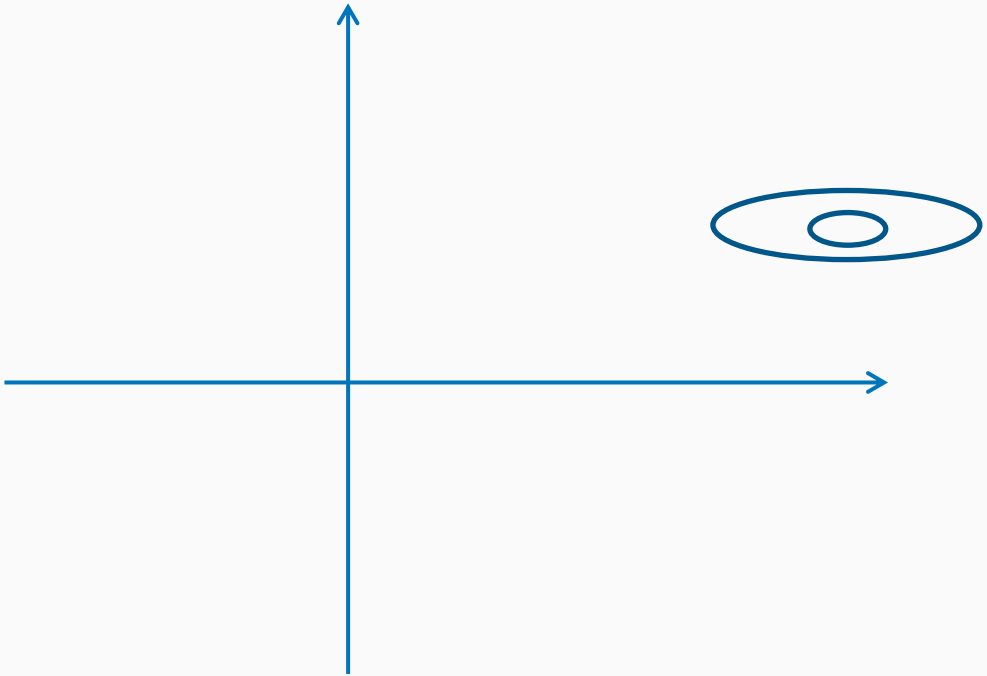
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



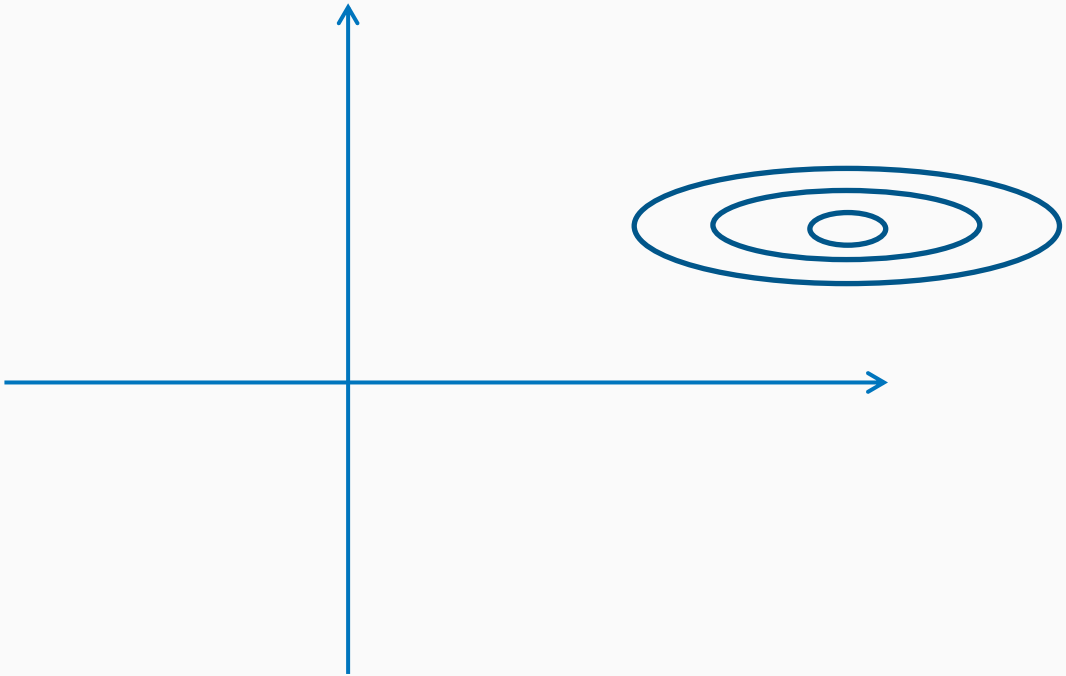
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



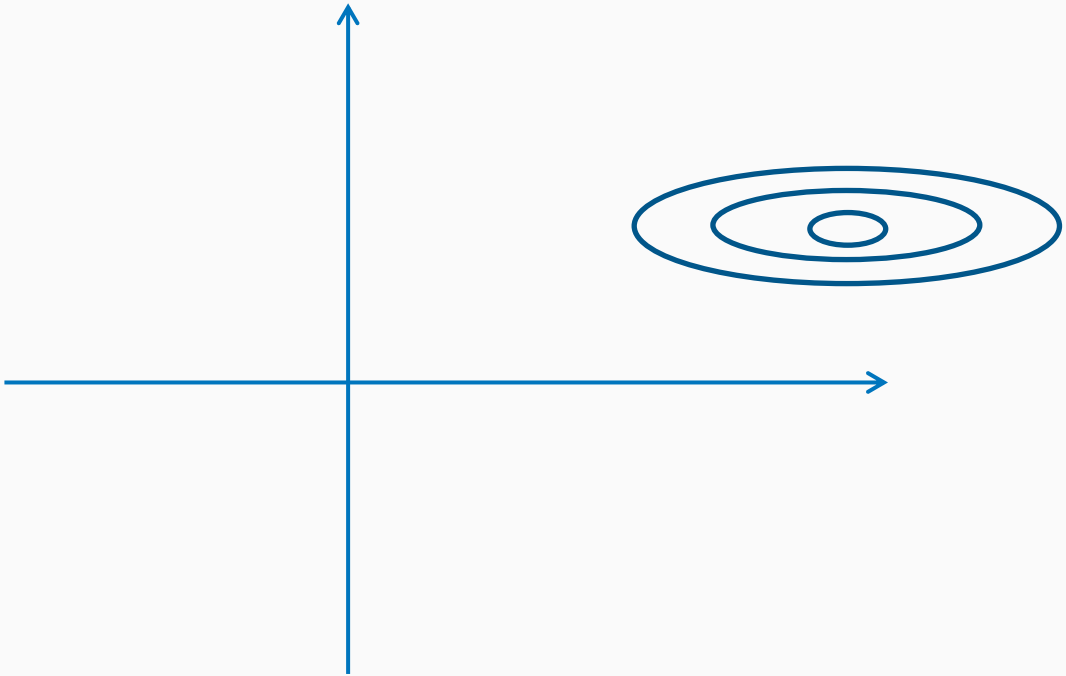
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



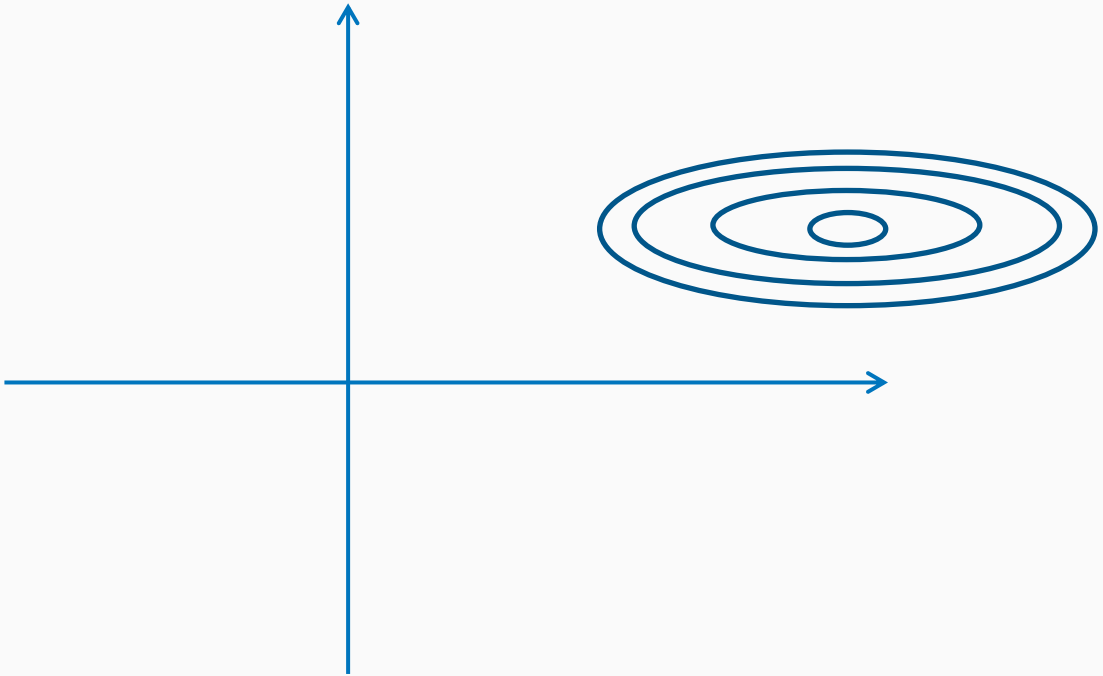
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



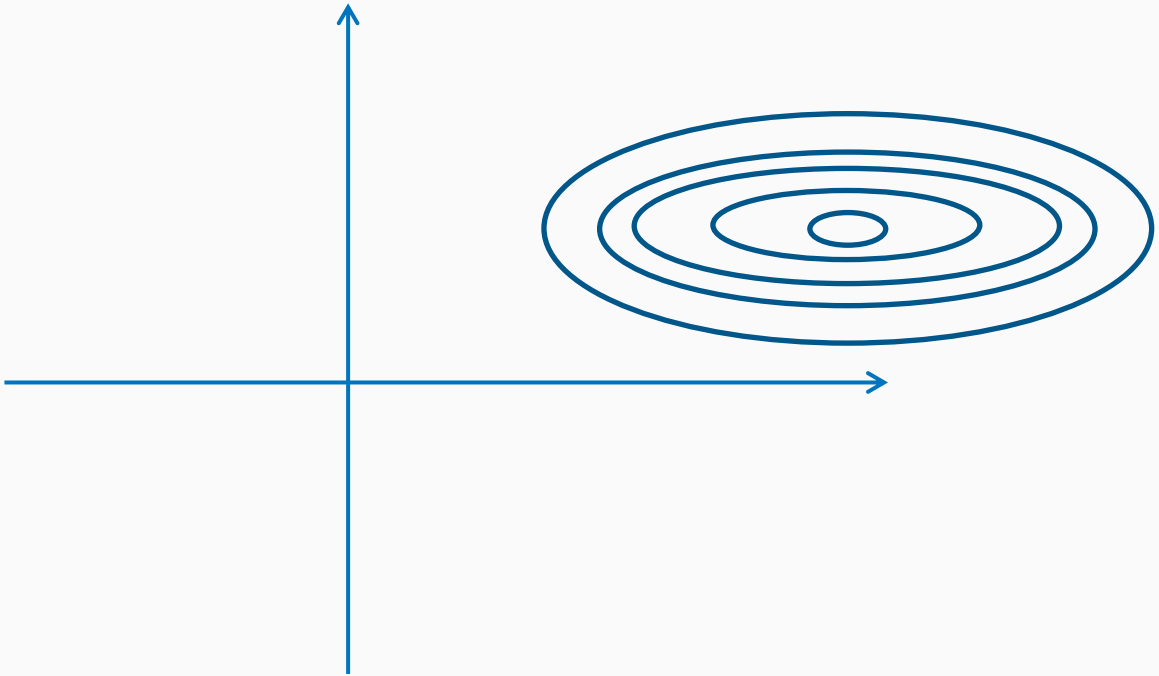
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



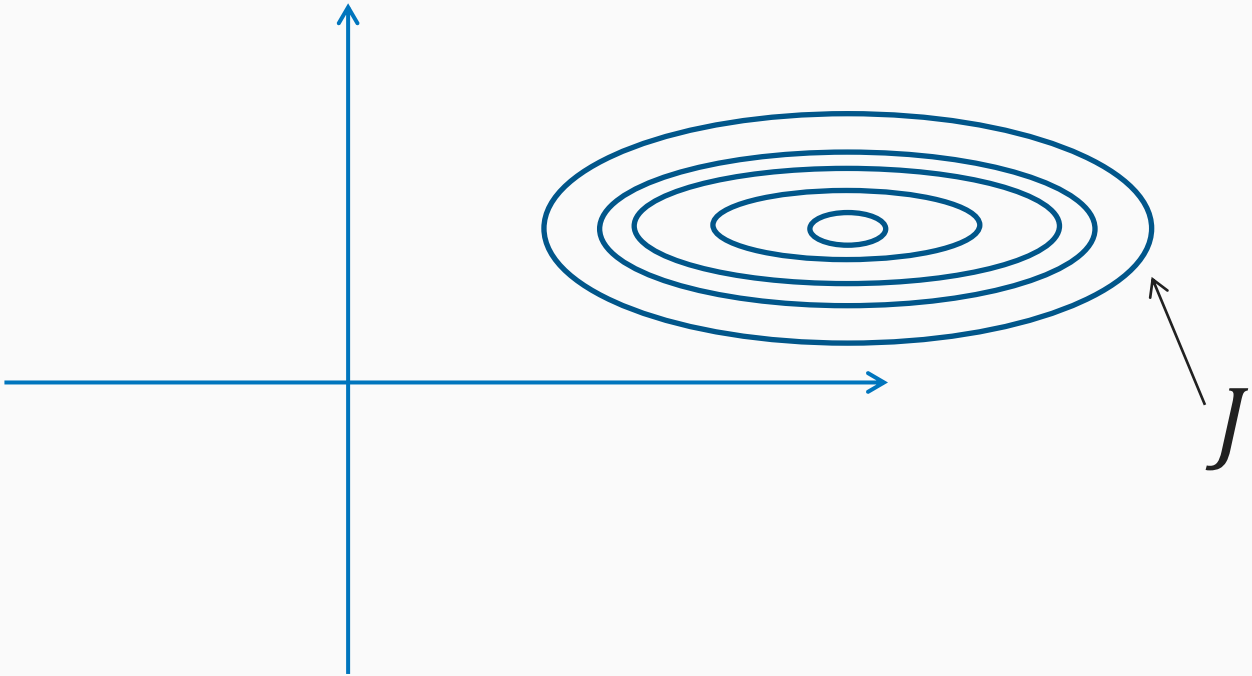
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



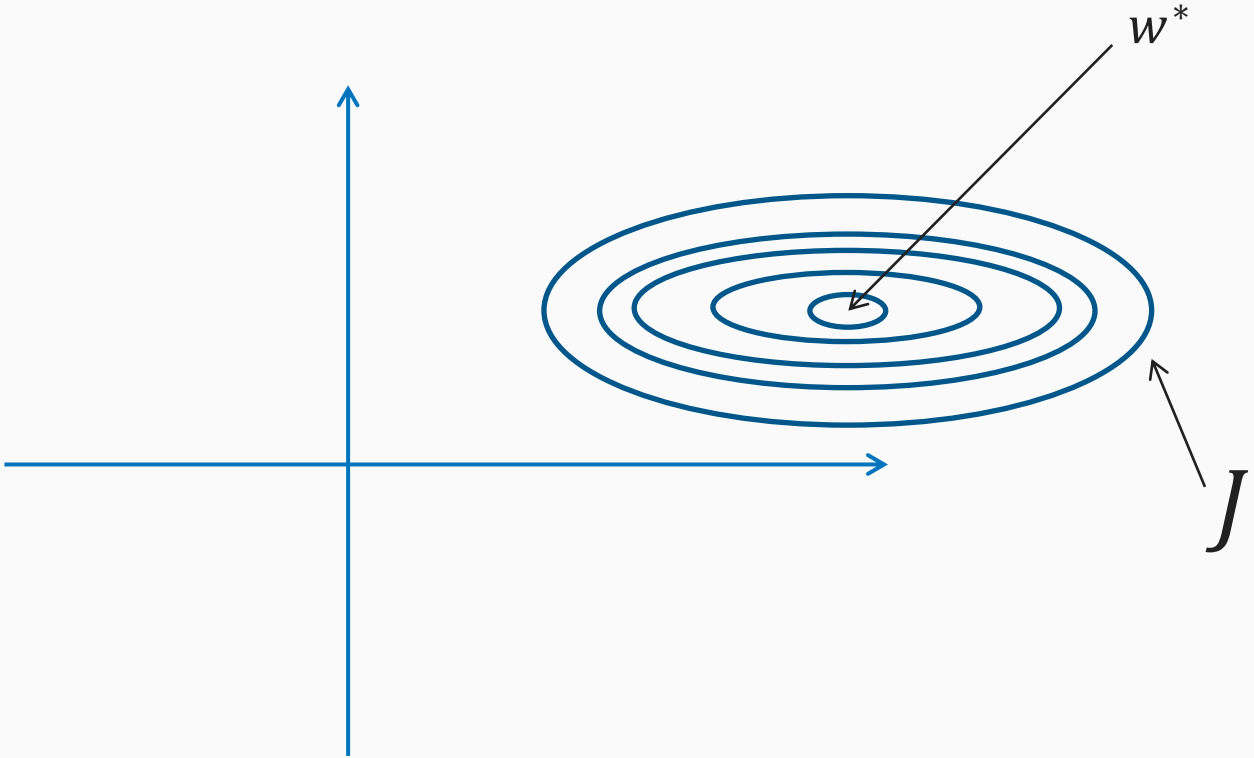
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



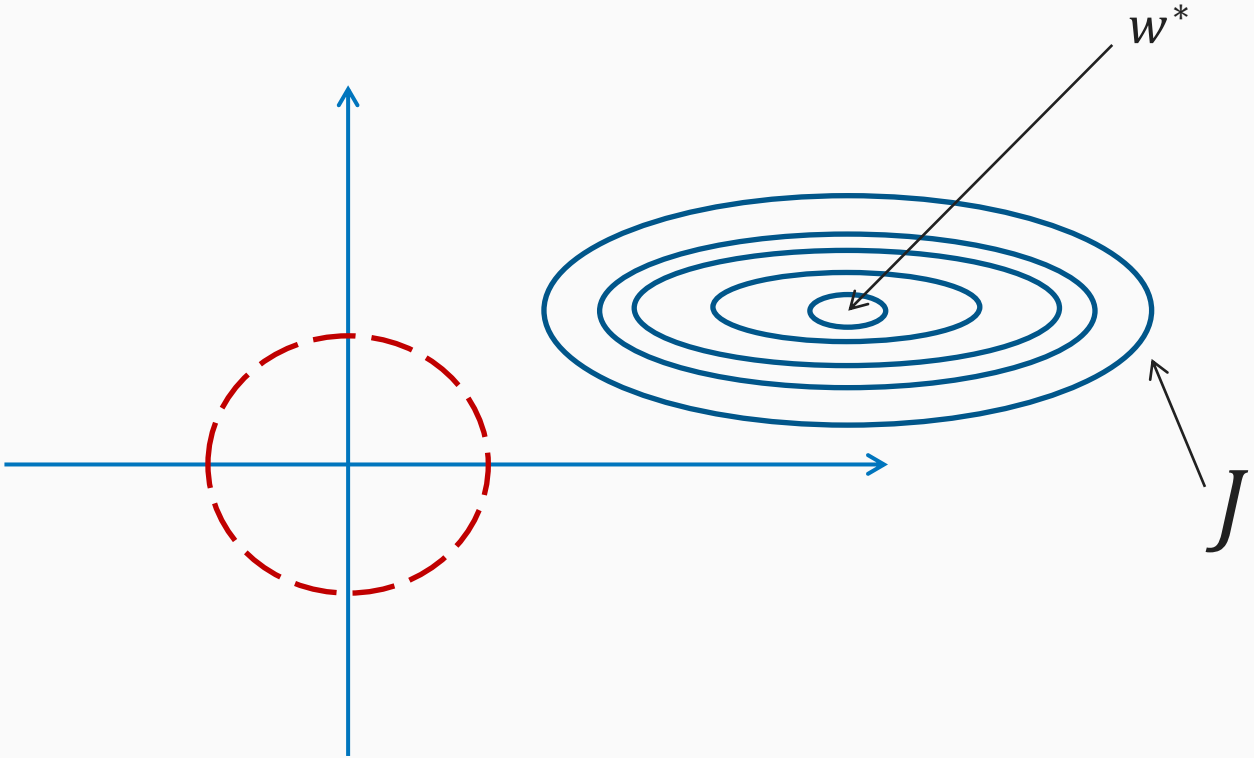
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



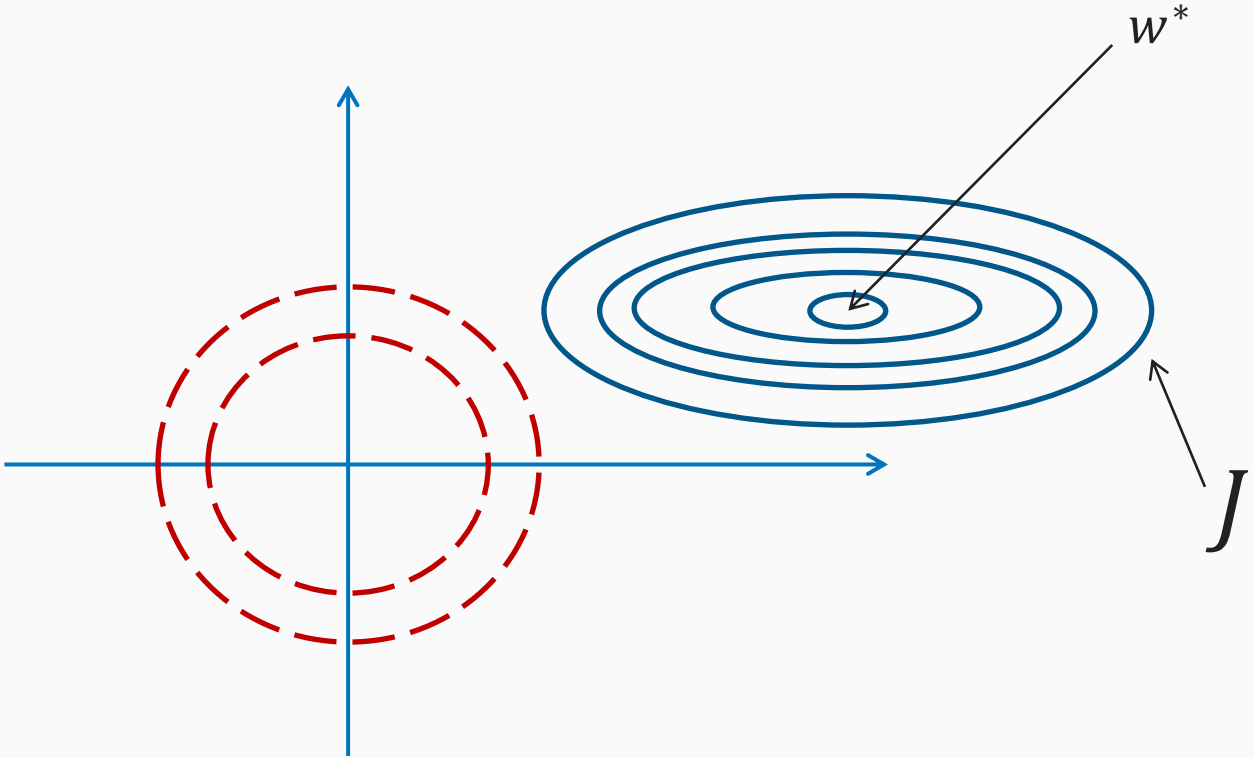
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



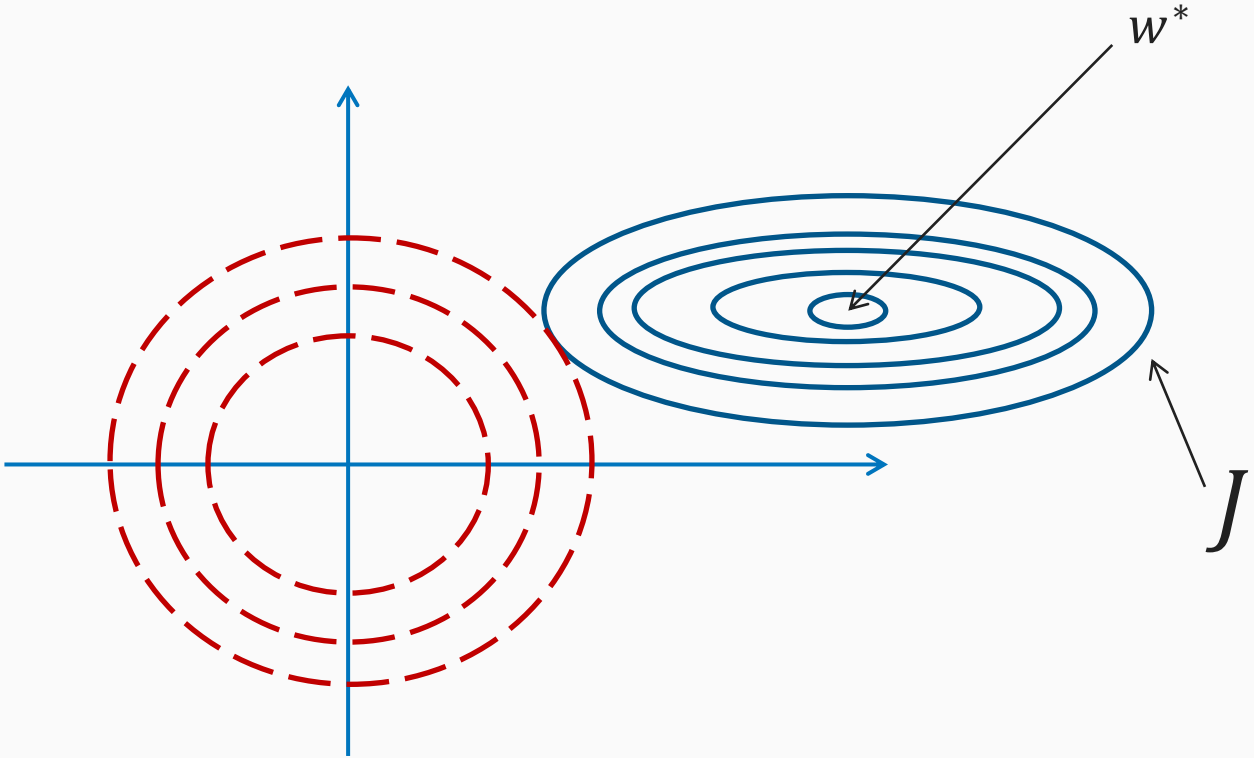
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



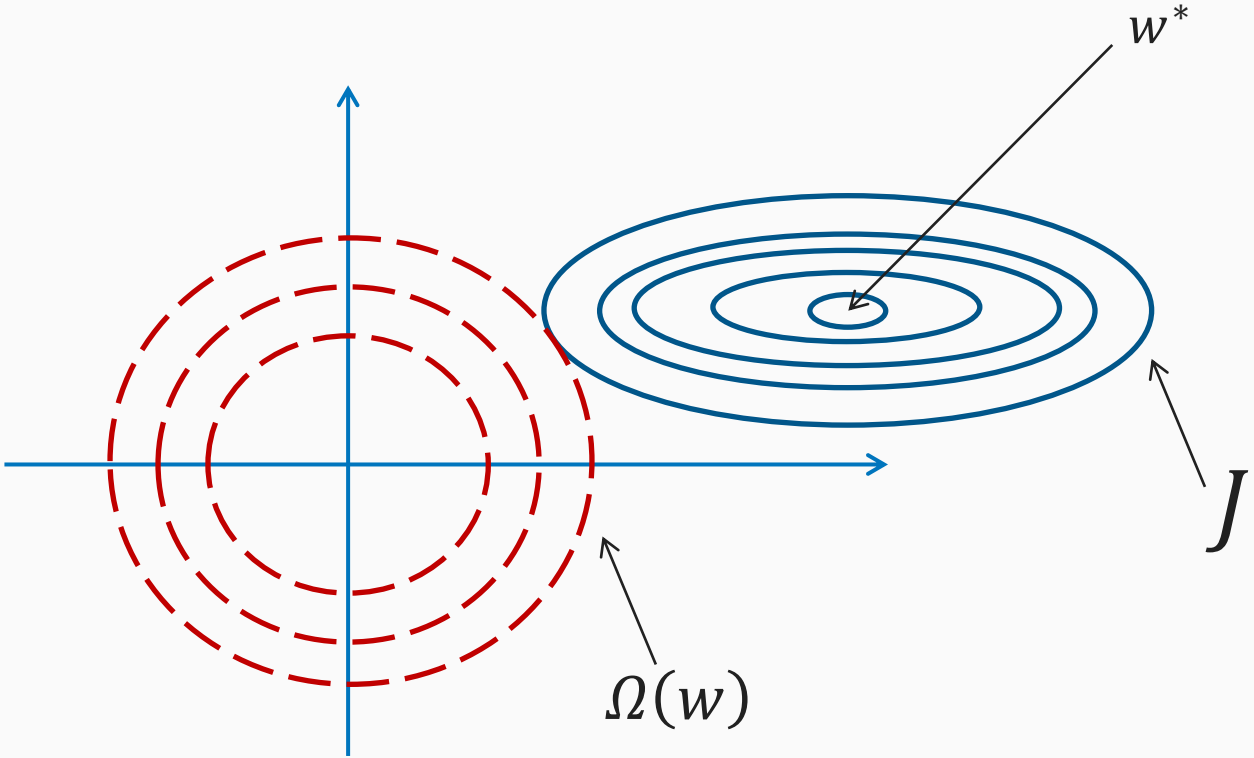
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



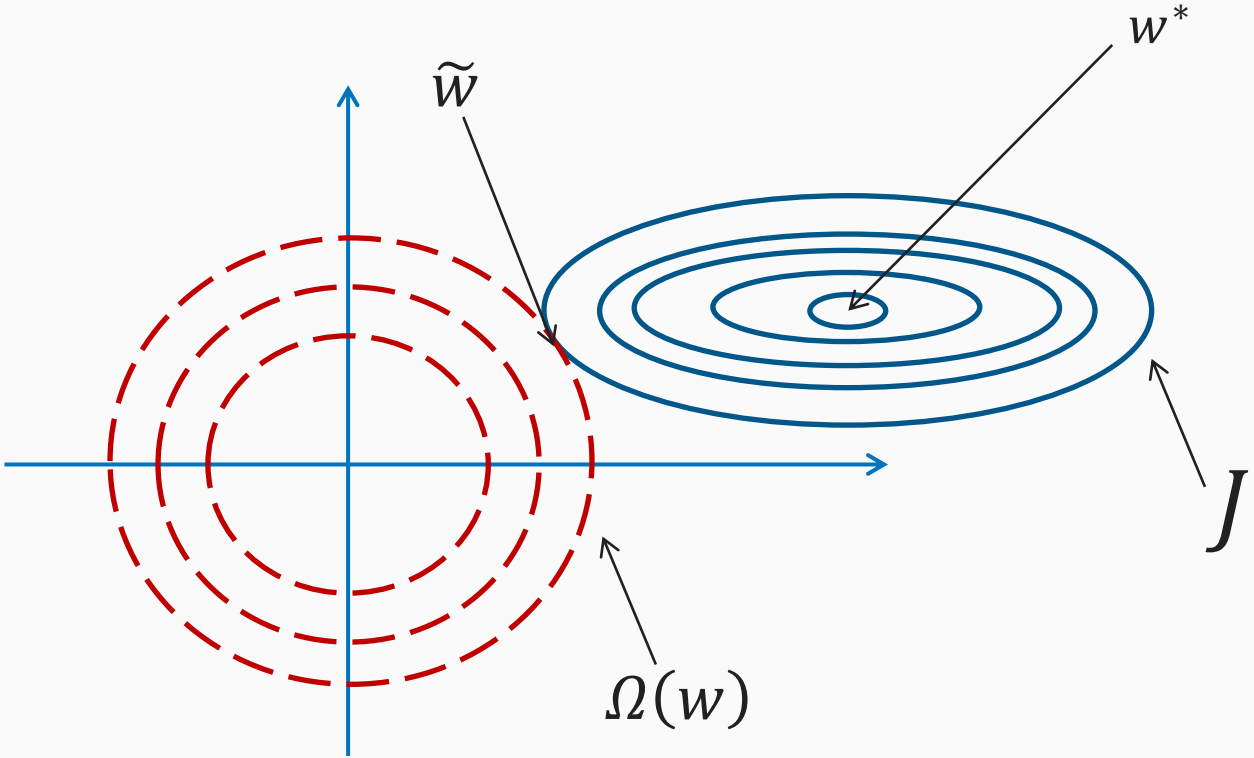
If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization

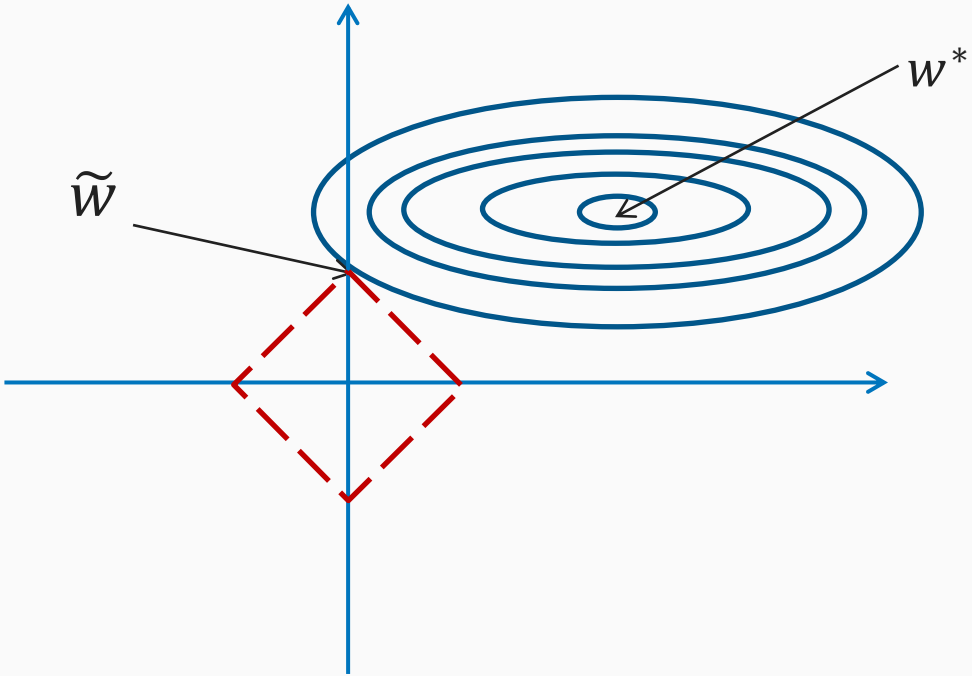


If $\Omega(w) = \alpha \|w\|_2^2$ the regularization is called L^2 regularization



If $\Omega(w) = \alpha\|w\|_1$ the regularization is called L^2 regularization

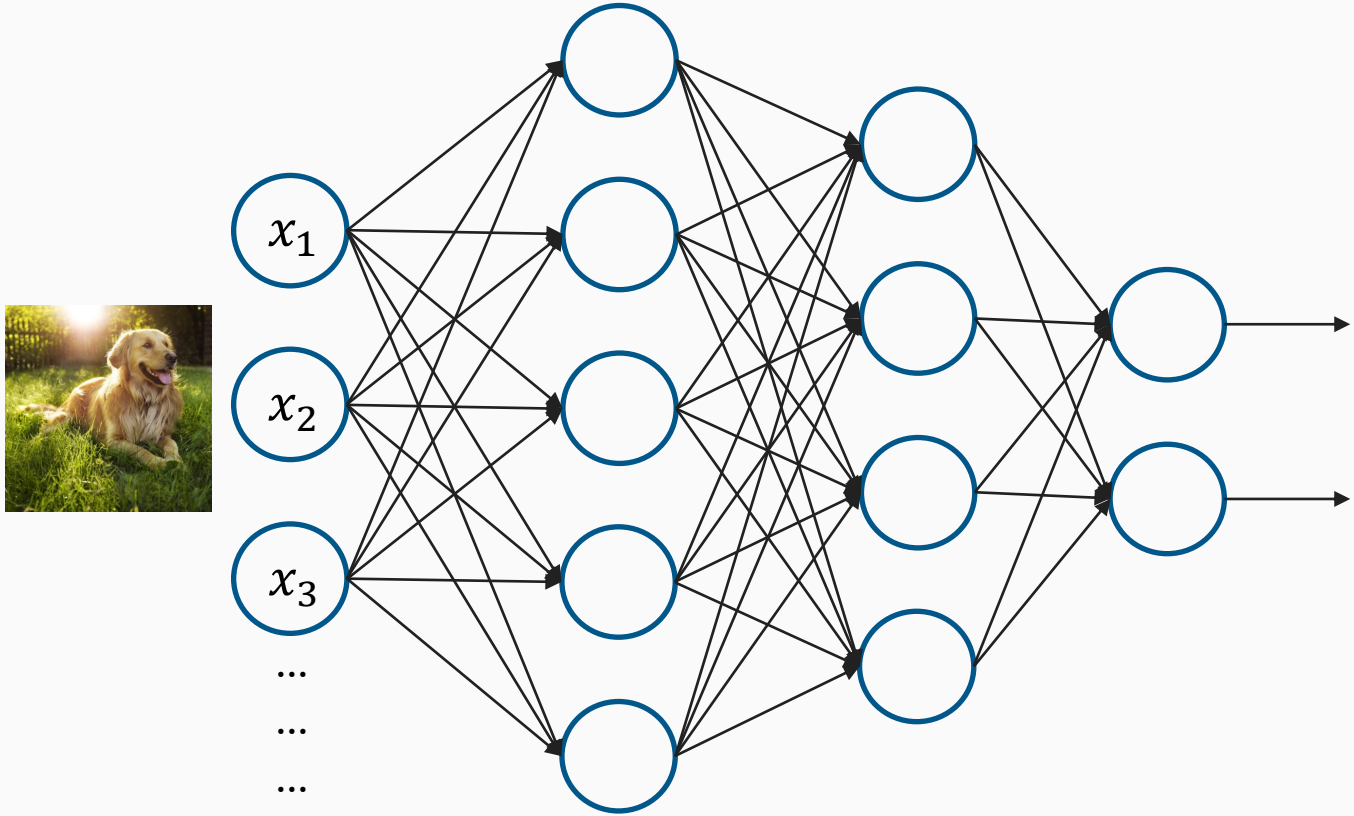
If $\Omega(w) = \alpha\|w\|_1$ the regularization is called L^2 regularization



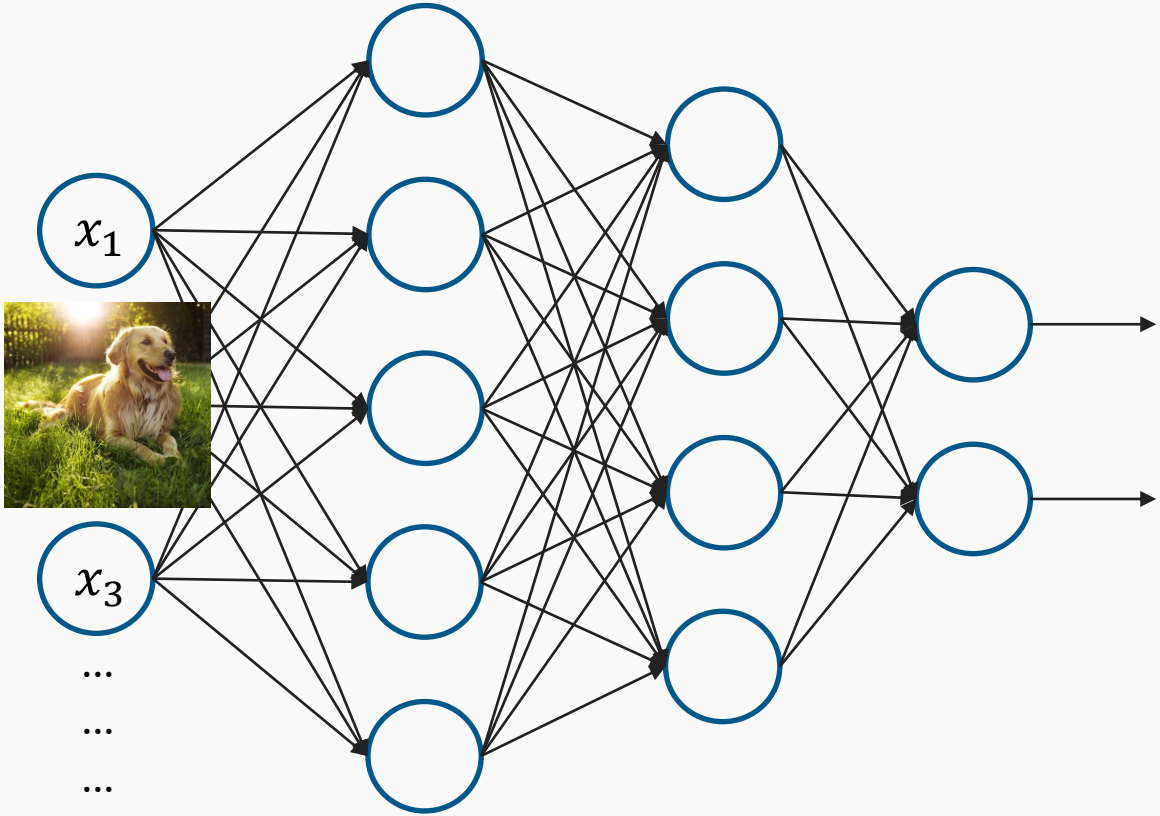
Dataset augmentation

Dataset augmentation

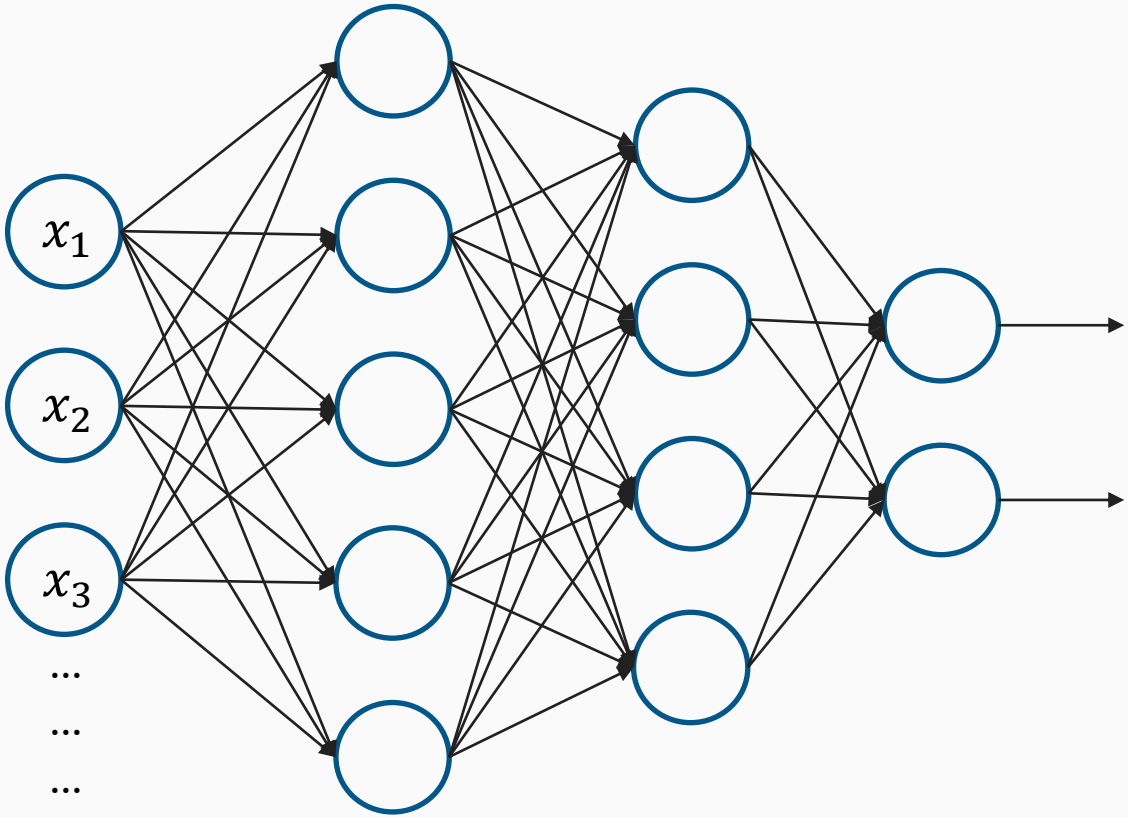
Dataset augmentation



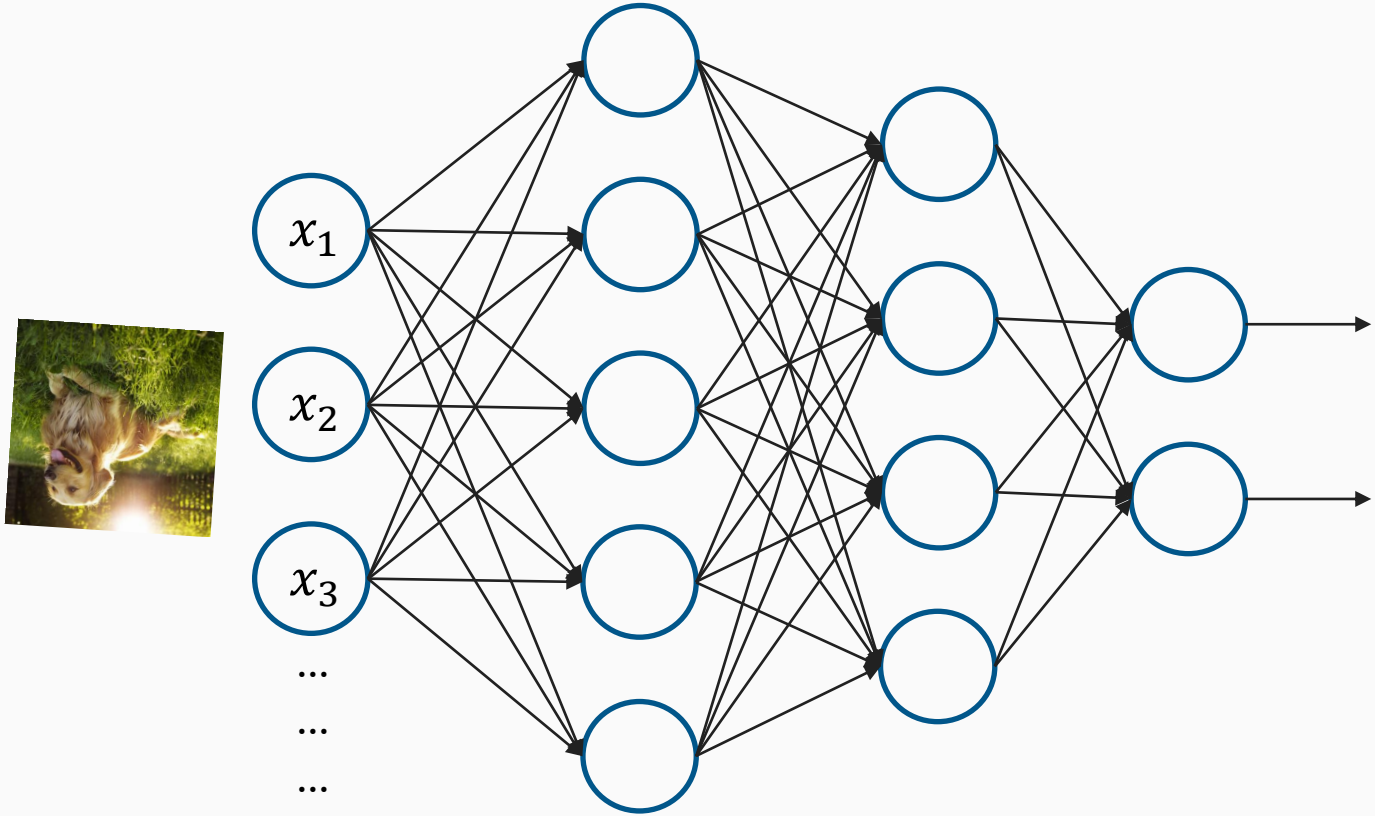
Dataset augmentation



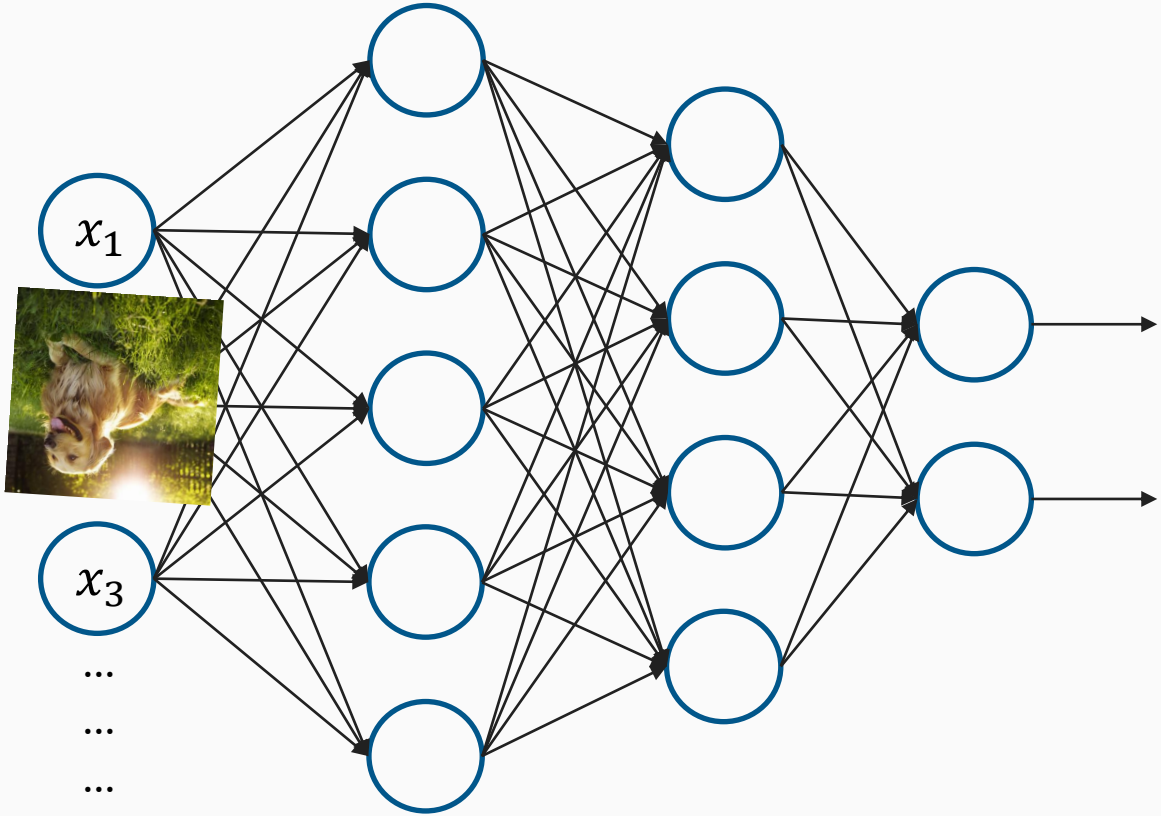
Dataset augmentation



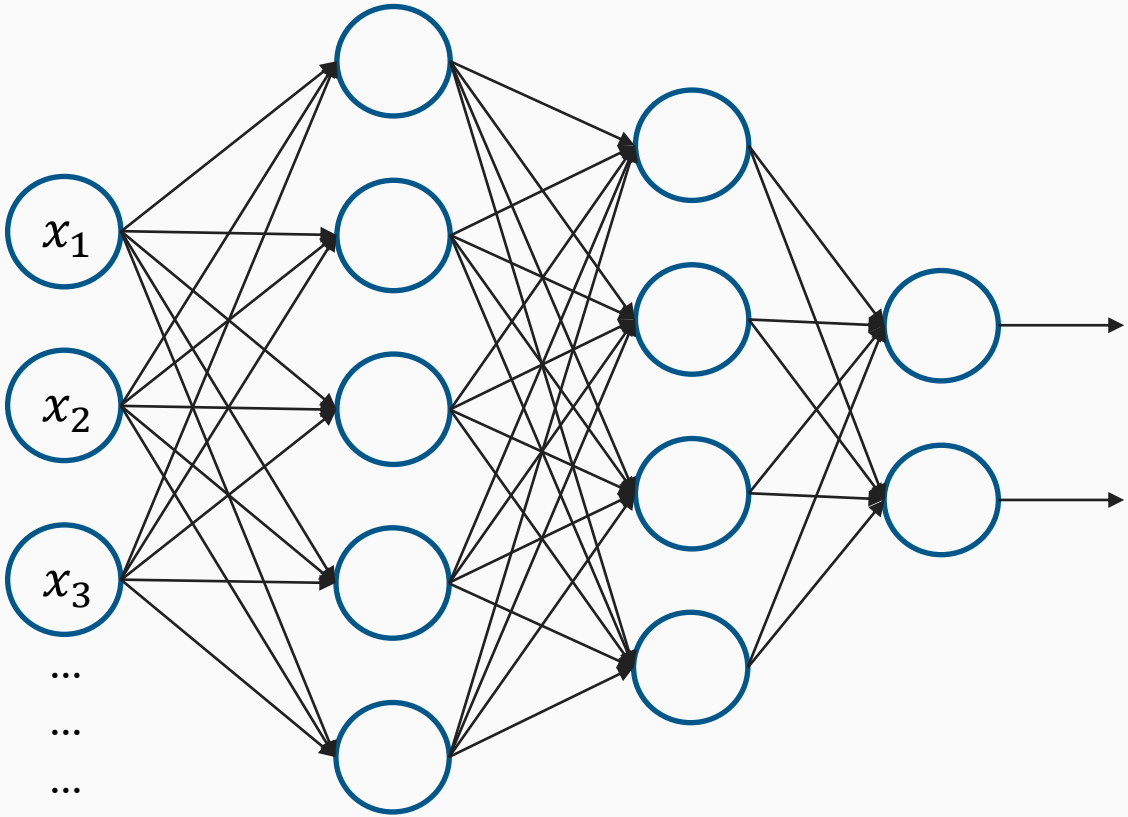
Dataset augmentation



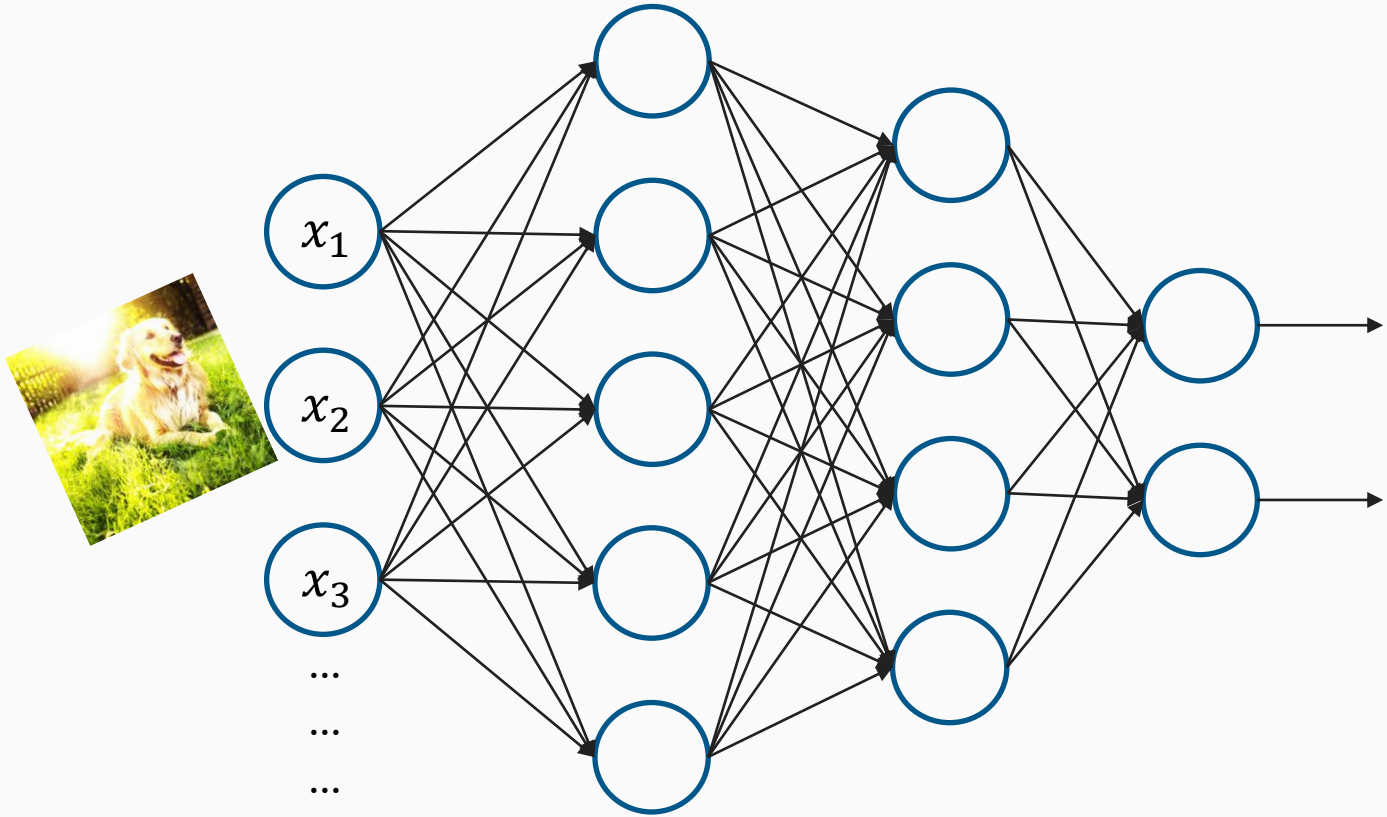
Dataset augmentation



Dataset augmentation



Dataset augmentation



Early stopping

Early stopping

We stop training when we reach a minimum

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

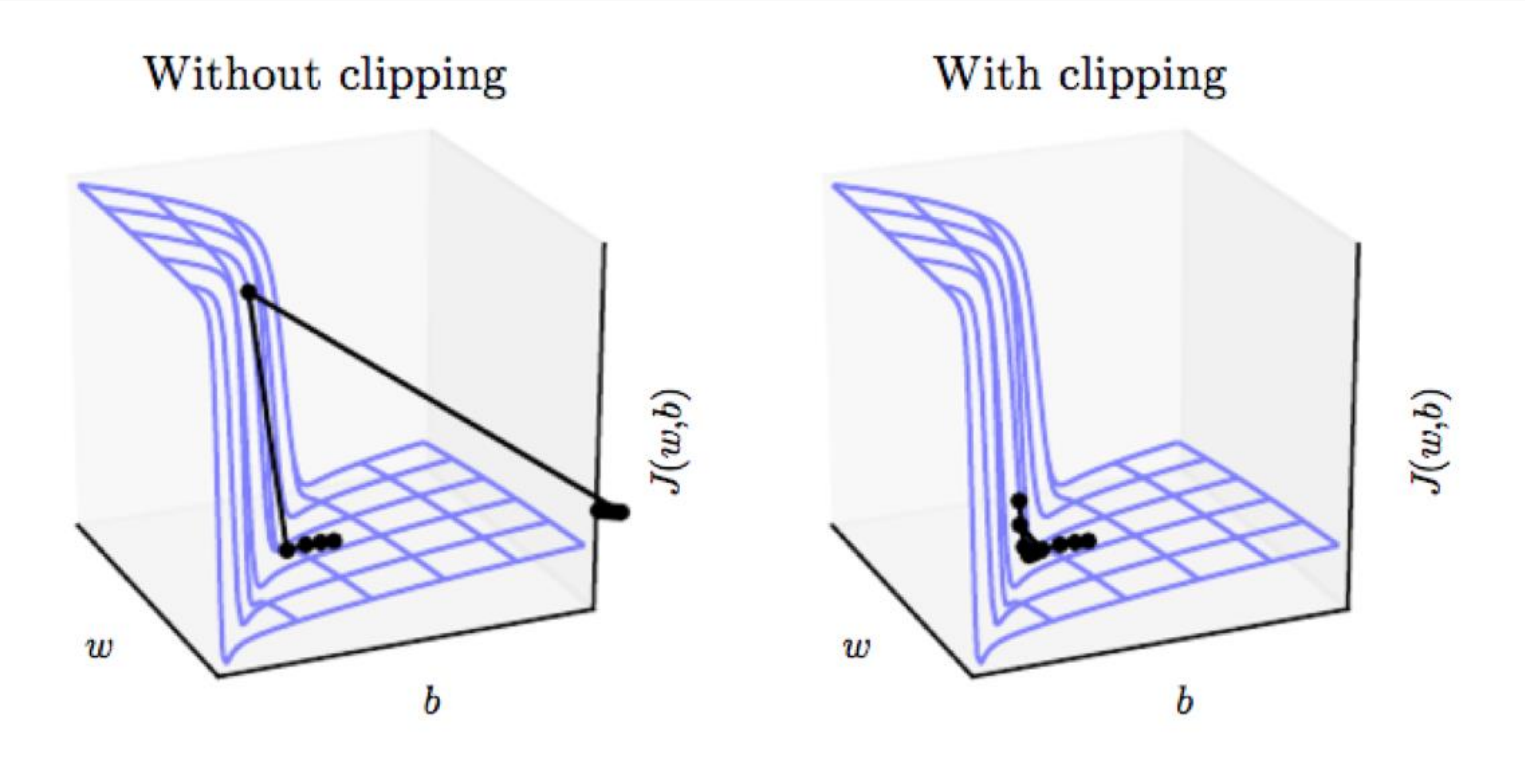
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Gradient clipping



PyTorch

PyTorch is an open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

Big companies uses PyTorch

OpenAI

Meta

Microsoft

TESLA

TOYOTA RESEARCH INSTITUTE

MITSUBISHI ELECTRIC

Genentech
A Member of the Roche Group

airbnb

QUALCOMM®

Why PyTorch ?

- Easy to learn
- dynamic computation graph
- Faster than Tensorflow
- Flexible
- Pythonic ☺
- Supports GPU and CPU
- Easy to debug
- Strong Community
- ....

How to install PyTorch ?

Go to https://pytorch.org/ and select the option that suits you the best:

PyTorch Build	Stable (1.10.2)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 10.2	CUDA 11.3	~~ROCm 4.2 (beta)~~	CPU
Run this Command:	`conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch`			

Previous versions of PyTorch >

The basic data type of PyTorch is the Tensor:

```
Entrée [1]:
1 import torch

Entrée [2]:
1 my_tensor = torch.tensor([[2, 4],[1, 5]])

Entrée [3]:
1 my_tensor

Out[3]: tensor([[2, 4],
                [1, 5]])
```

Basic operations :

Addition:

Entrée [1]:
```python
import torch
```

Entrée [6]:
```python
my_tensor_1 = torch.tensor([[11, 4.4],[1, 5]])
```

Entrée [4]:
```python
my_tensor_2 = torch.tensor([[2, 24],[11, 5]])
```

Entrée [7]:
```python
my_tensor_1 + my_tensor_2
```

Out[7]:
```
tensor([[13.0000, 28.4000],
        [12.0000, 10.0000]])
```

Entrée [8]:
```python
torch.add(my_tensor_1, my_tensor_2)
```

Out[8]:
```
tensor([[13.0000, 28.4000],
        [12.0000, 10.0000]])
```

Basic operations :

Subtraction:

Entrée [1]:
```
import torch
```

Entrée [6]:
```
my_tensor_1 = torch.tensor([[11, 4.4],[1, 5]])
```

Entrée [4]:
```
my_tensor_2 = torch.tensor([[2, 24],[11, 5]])
```

Entrée [9]:
```
my_tensor_1 - my_tensor_2
```

Out[9]:
```
tensor([[  9.0000, -19.6000],
        [-10.0000,   0.0000]])
```

Entrée [11]:
```
torch.subtract(my_tensor_1, my_tensor_2)
```

Out[11]:
```
tensor([[  9.0000, -19.6000],
        [-10.0000,   0.0000]])
```

Basic operations :

multiplication:

Entrée [1]:
```python
import torch
```

Entrée [6]:
```python
my_tensor_1 = torch.tensor([[11, 4.4],[1, 5]])
```

Entrée [4]:
```python
my_tensor_2 = torch.tensor([[2, 24],[11, 5]])
```

Entrée [12]:
```python
my_tensor_1 * my_tensor_2
```

Out[12]:
```
tensor([[ 22.0000, 105.6000],
        [ 11.0000,  25.0000]])
```

Entrée [13]:
```python
torch.multiply(my_tensor_1, my_tensor_2)
```

Out[13]:
```
tensor([[ 22.0000, 105.6000],
        [ 11.0000,  25.0000]])
```

Basic operations :

division:

Entrée [1]:
```python
import torch
```

Entrée [6]:
```python
my_tensor_1 = torch.tensor([[11, 4.4],[1, 5]])
```

Entrée [4]:
```python
my_tensor_2 = torch.tensor([[2, 24],[11, 5]])
```

Entrée [14]:
```python
my_tensor_1 / my_tensor_2
```

Out[14]:
```
tensor([[5.5000, 0.1833],
        [0.0909, 1.0000]])
```

Entrée [16]:
```python
torch.divide(my_tensor_1, my_tensor_2)
```

Out[16]:
```
tensor([[5.5000, 0.1833],
        [0.0909, 1.0000]])
```

Basic operations :

And many moooooooore. To check it out :
https://pytorch.org/docs/stable/tensors.html

Let’s use the boring MNIST dataset ☹ to create our first network using PyTorch

The idea is to create a neural net that will see all the images below, and then we will give it a new image and it should return the class of the image i.e 0, 1, 2, 3, 4, 5, 6, 7, 8 ,9

First we need to download the dataset:

To do that, go to the http://yann.lecun.com/exdb/mnist

Next we need to import some things:

```
import warnings
warnings.filterwarnings("ignore")
import torch
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision.io import read_image
import torch.nn as nn
from torch import optim
import os
import glob
from torchvision import transforms
```

We will create a class that inherits from Dataset to provide it with more functionalities:

```python
import warnings
warnings.filterwarnings("ignore")
import torch
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision.io import read_image
import torch.nn as nn
from torch import optim
import os
import glob
from torchvision import transforms
```

We will create a class that inherits from Dataset to provide it with more functionalities:

```
class Data(Dataset):
    def __init__(self, transform = None):
        super().__init__()
        self.images = glob.glob(r'../lecture4/code/train/*/*')
        self.labels = [int(image.split(os.path.sep)[-2]) for image in self.images]
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self,idx):
        img = read_image(self.images[idx])
        label = self.labels[idx]
        if self.transform:
            img = self.transform(img)
        return img/255.0, label
```

A PyTorch dataset class must have at least 3 methods : __init__, __len__, __getitem__:

```
class Data(Dataset):
    def __init__(self, transform = None):
        super().__init__()
        self.images = glob.glob(r'../lecture4/code/train/*/*')
        self.labels = [int(image.split(os.path.sep)[-2]) for image in self.images]
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self,idx):
        img = read_image(self.images[idx])
        label = self.labels[idx]
        if self.transform:
            img = self.transform(img)
        return img/255.0, label
```

We usually use __init__ to provide the paths to our images and create their labels without loading them (images):

```
class Data(Dataset):
    def __init__(self, transform = None):
        super().__init__()
        self.images = glob.glob(r'../lecture4/code/train/*/*')
        self.labels = [int(image.split(os.path.sep)[-2]) for image in self.images]
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self,idx):
        img = read_image(self.images[idx])
        label = self.labels[idx]
        if self.transform:
            img = self.transform(img)
        return img/255.0, label
```

__len__: to determine the shape of our dataset:

```
class Data(Dataset):
    def __init__(self, transform = None):
        super().__init__()
        self.images = glob.glob(r'../lecture4/code/train/*/*')
        self.labels = [int(image.split(os.path.sep)[-2]) for image in self.images]
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self,idx):
        img = read_image(self.images[idx])
        label = self.labels[idx]
        if self.transform:
            img = self.transform(img)
        return img/255.0, label
```

__getitem__: to fetch the data when indexing e.g Data[0][0] will return the first image in our dataset:

```
class Data(Dataset):
    def __init__(self, transform = None):
        super().__init__()
        self.images = glob.glob(r'../lecture4/code/train/*/*')
        self.labels = [int(image.split(os.path.sep)[-2]) for image in self.images]
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self,idx):
        img = read_image(self.images[idx])
        label = self.labels[idx]
        if self.transform:
            img = self.transform(img)
        return img/255.0, label
```

After that, we specify our transformations to apply on images and split the dataset into training, testing and validation:

```
tr = transforms.Compose([
    transforms.Resize((28,28)),
    transforms.Lambda(lambda x : torch.flatten(x))
])
```

```
data = Data(tr)
```

```
data[0][0].shape
```

```
torch.Size([784])
```

```
train_ds ,test_ds, val_ds = random_split(data,[33600,4200,4200])
```

After splitting the dataset, we use the DataLoader class to create batches for us to be able to iterate through them. This is good for memory efficiency and also it uses data augmentation under the hood:

```
train_dl = DataLoader(train_ds, batch_size = 16, drop_last=True, shuffle = True)
test_dl = DataLoader(test_ds, batch_size = 16, drop_last=True, shuffle = False)
val_dl = DataLoader(val_ds, batch_size = 16, drop_last=True, shuffle = False)
```

And finally, we can create our first network by creating a class that inherits from the nn.Module:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear_1 = nn.Linear(784, 1000)
        self.linear_2 = nn.Linear(1000, 1000)
        self.linear_3 = nn.Linear(1000, 200)
        self.linear_4 = nn.Linear(200, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.softmax = nn.Softmax()


    def forward(self,x):
        x = self.linear_1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.linear_2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.linear_3(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.linear_4(x)
        x = self.softmax(x)
        return x
```

We create our object net, we move the neural net to the GPU and we define the cost, epochs and optimizer:

```
net = Net()
```

```
net.to('cuda');
```

```
cost = nn.CrossEntropyLoss()
```

```
optimizer = optim.Adam(net.parameters(), lr = 0.001)
```

```
epochs = 30
```

The last step is to implement a training loop :

```
cost_train = []
accuracy = []

cost_val = []
accuracy_val = []

for epoch in range(1, 1+epochs):
    net.train()
    train_cost = 0.0
    num_correct = 0.0
    num_examples = 0.0
    for x, y in train_dl:
        optimizer.zero_grad()
        x = x.to('cuda', dtype = torch.float)
        y = y.to('cuda', dtype = torch.long)
        pred = net(x)
        error = cost(pred, y)
        error.backward()
        optimizer.step()
        train_cost = train_cost + error.item()
        num_correct = pred.argmax(dim=1).eq(y).sum().item() + num_correct
        num_examples = y.size(0) + num_examples
    cost_train.append(train_cost)
    accuracy.append(num_correct/num_examples)

    with torch.no_grad():
        net.eval()
        val_cost = 0.0
        num_correct_val = 0.0
```

The last step is to implement a training loop :

```python
cost_train = []
accuracy = []

cost_val = []
accuracy_val = []

for epoch in range(1, 1+epochs):
    net.train()
    train_cost = 0.0
    num_correct = 0.0
    num_examples = 0.0
    for x, y in train_dl:
        optimizer.zero_grad()
        x = x.to('cuda', dtype = torch.float)
        y = y.to('cuda', dtype = torch.long)
        pred = net(x)
        error = cost(pred, y)
        error.backward()
        optimizer.step()
        train_cost = train_cost + error.item()
        num_correct = pred.argmax(dim=1).eq(y).sum().item() + num_correct
        num_examples = y.size(0) + num_examples
    cost_train.append(train_cost)
    accuracy.append(num_correct/num_examples)

    with torch.no_grad():
        net.eval()
        val_cost = 0.0
        num_correct_val = 0.0
```

The optimizer accumulates the derivatives. To not do that we use zero_grad

Forward pass

Compute the cost

Backward pass
(apply backpropagation)

Update the weights and biases

It is used to not compute the derivatives of the tensors

It is used to turn off the dropout layers, Batchnormalization ,...

The last step is to implement a training loop :

```
    accuracy.append(num_correct/num_examples)

    with torch.no_grad():
        net.eval()
        val_cost = 0.0
        num_correct_val = 0.0
        num_examples_val = 0.0
        for x, y in val_dl:
            x = x.to('cuda', dtype = torch.float)
            y = y.to('cuda', dtype = torch.long)
            pred = net(x)
            error = cost(pred, y)
            val_cost = val_cost + error.item()
            num_correct_val = pred.argmax(dim=1).eq(y).sum().item() + num_correct_val
            num_examples_val = y.size(0) + num_examples_val
        cost_val.append(val_cost)
        accuracy.append(num_correct_val/num_examples_val)

    print(f'epoch : {epoch} , training cost : {round(train_cost/len(train_dl),2)} , training accuracy : {round(num
    print('*****************************************************************************************************
    print('\n')
```

And congrats for implementing your first neural network using PyTorch

In the next lecture we will train a neural net to drive a car (self driving car using behavioral cloning) using PyTorch.

end