Introduction to Neural Networks

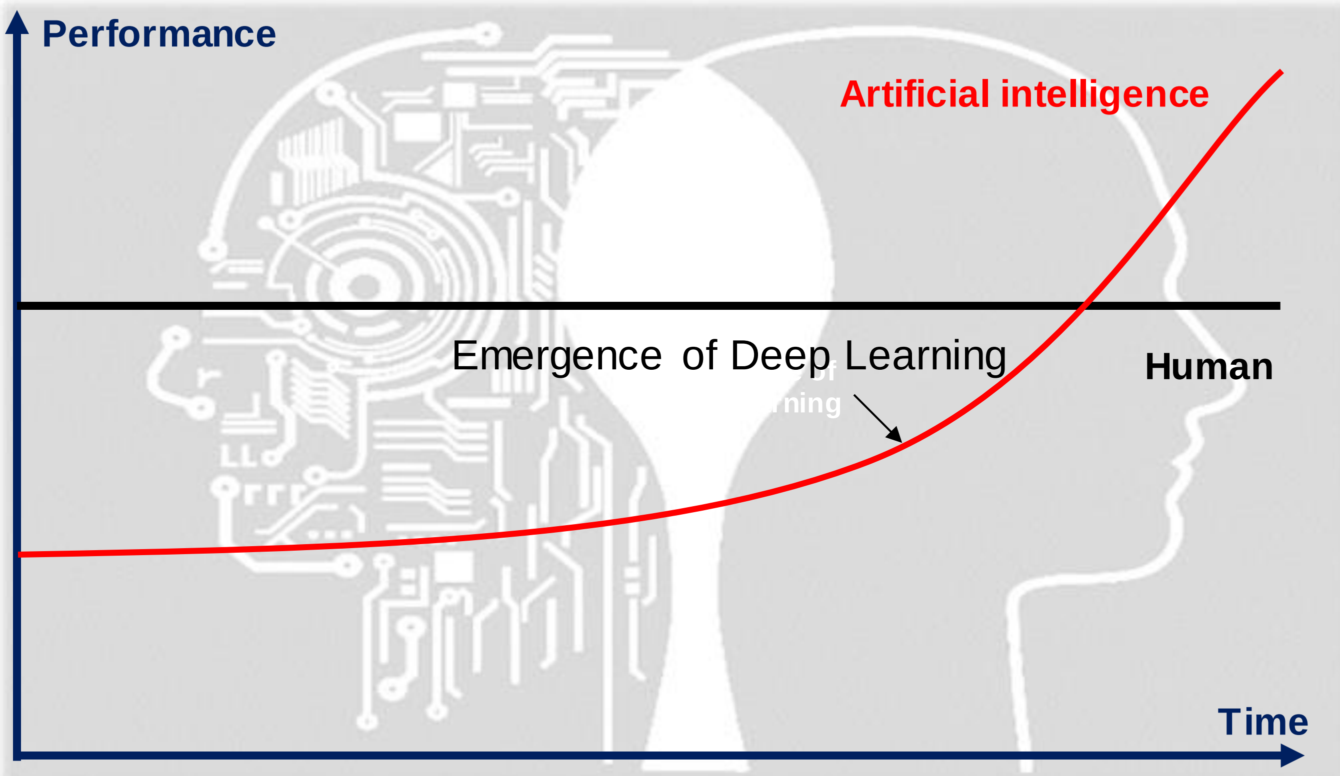
Philipp Seegerer and Seul-Ki Yeom
Technische Universität Berlin - Machine Learning Group

2019 Beginners Workshop Machine Learning

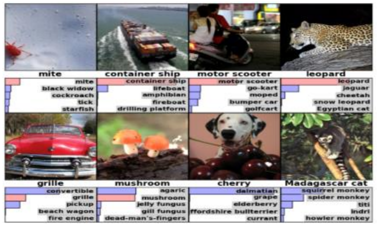
Contents

- History of NN
- Rosenblatt's Perceptron
  - Cost (= Loss) function
  - Gradient descent algorithm
- Multi-Layer Perceptron (MLP)
  - Activation function
  - Backpropagation

Emergence of Deep Learning



IBM Watson



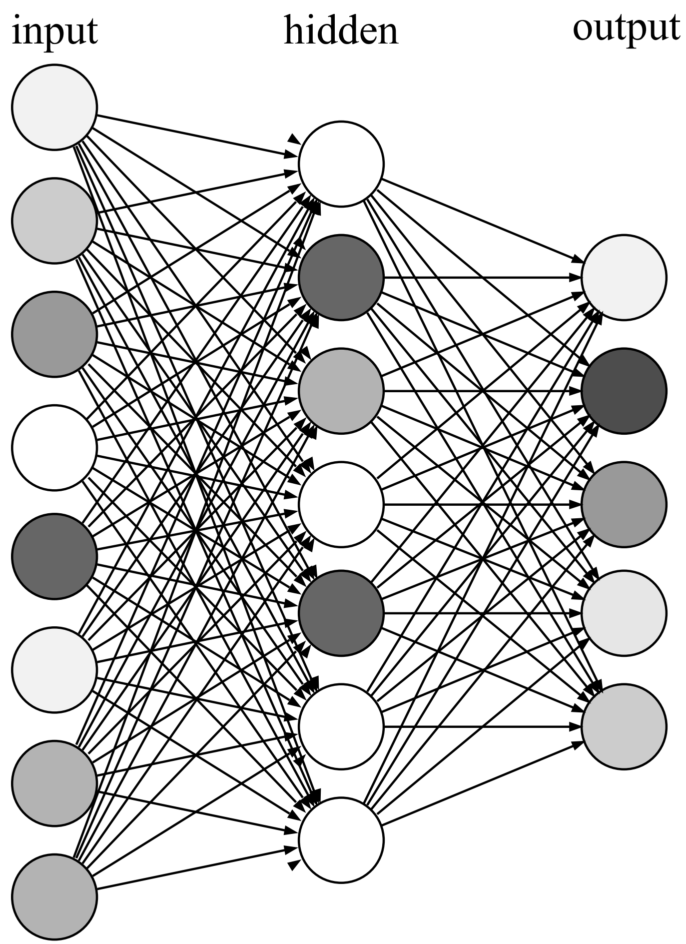
ImageNet


AlphaGo

- The technology of AI has been remarkably developed and now has surpassed human intelligence

(Artificial) Neural Network

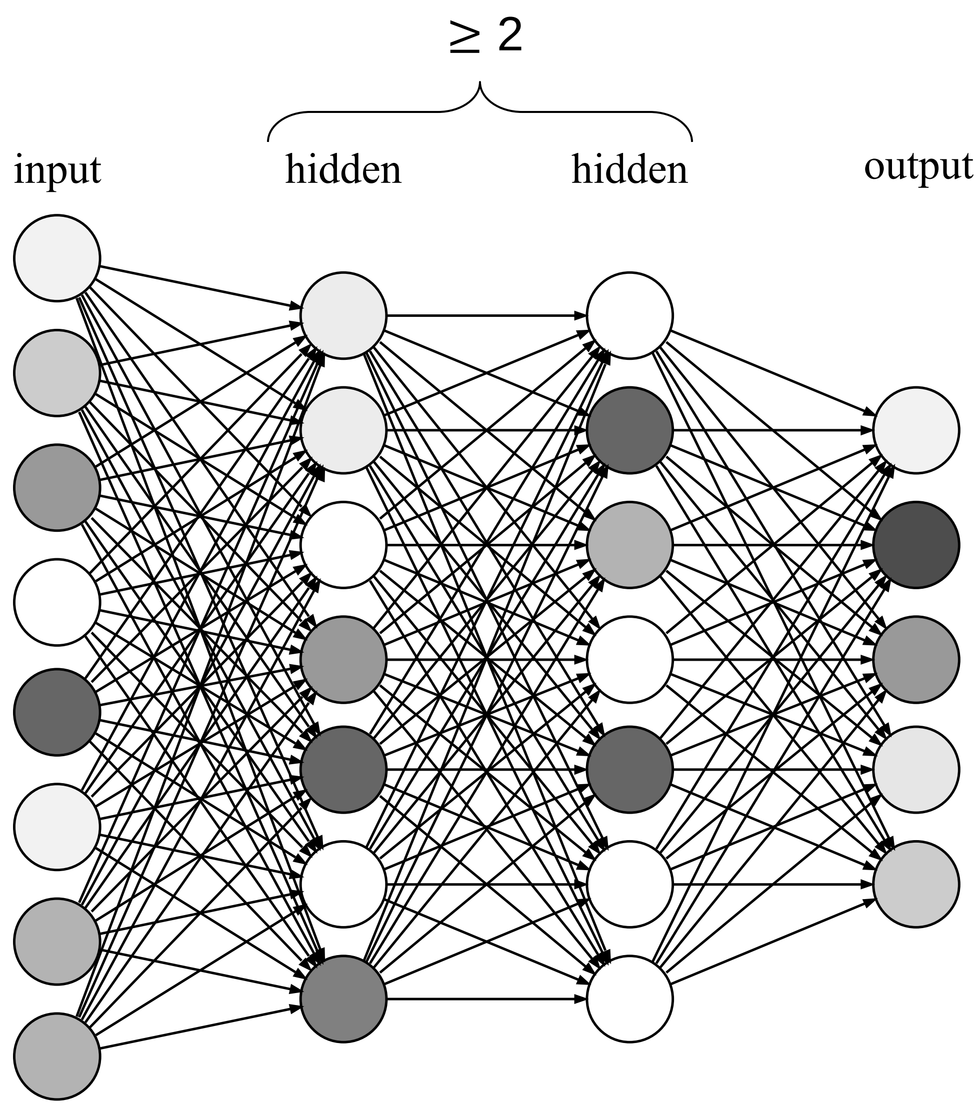
- Deep learning is one of the ML research tool which uses NN
- Pieces of math !



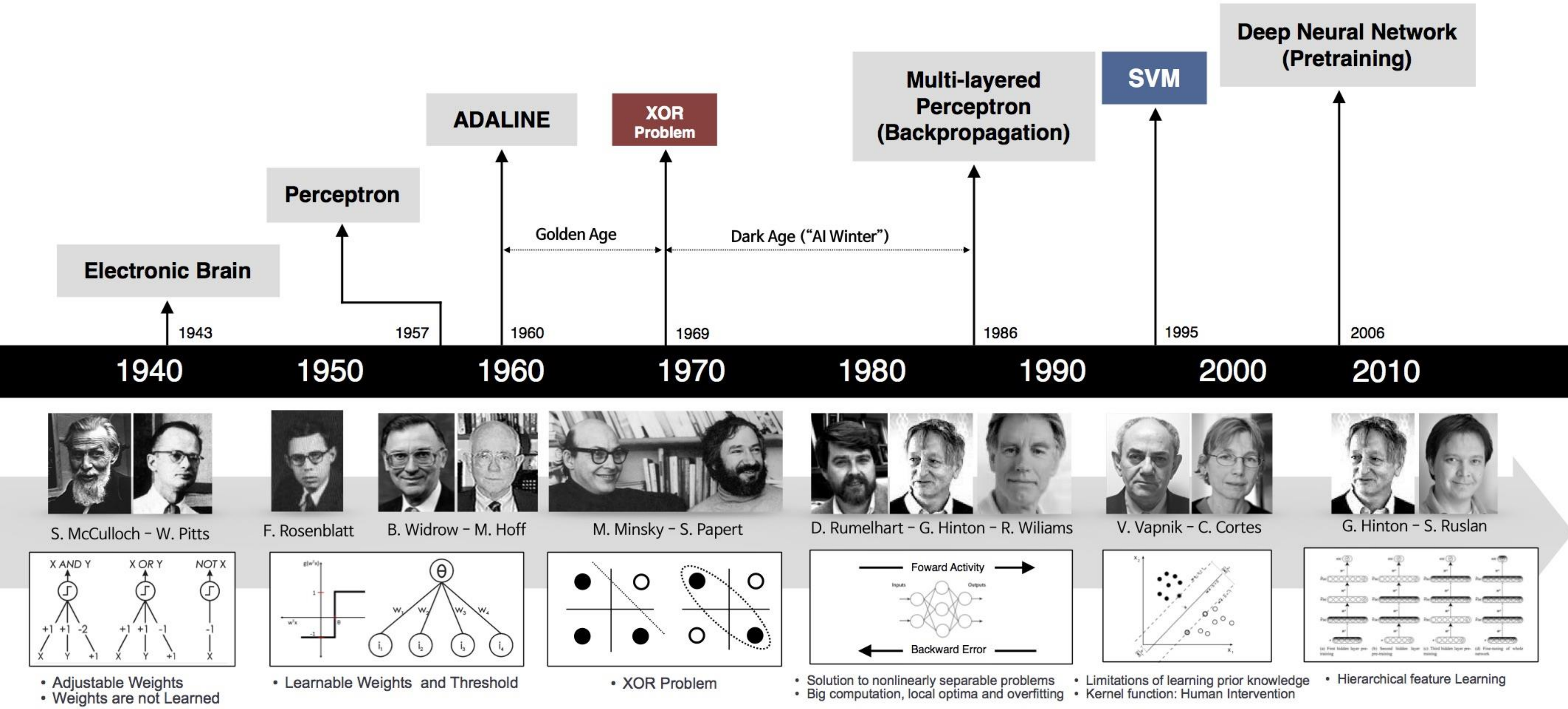
Multi-layer Perceptron

Deep Neural Network

- # of hidden layer ≥ 2



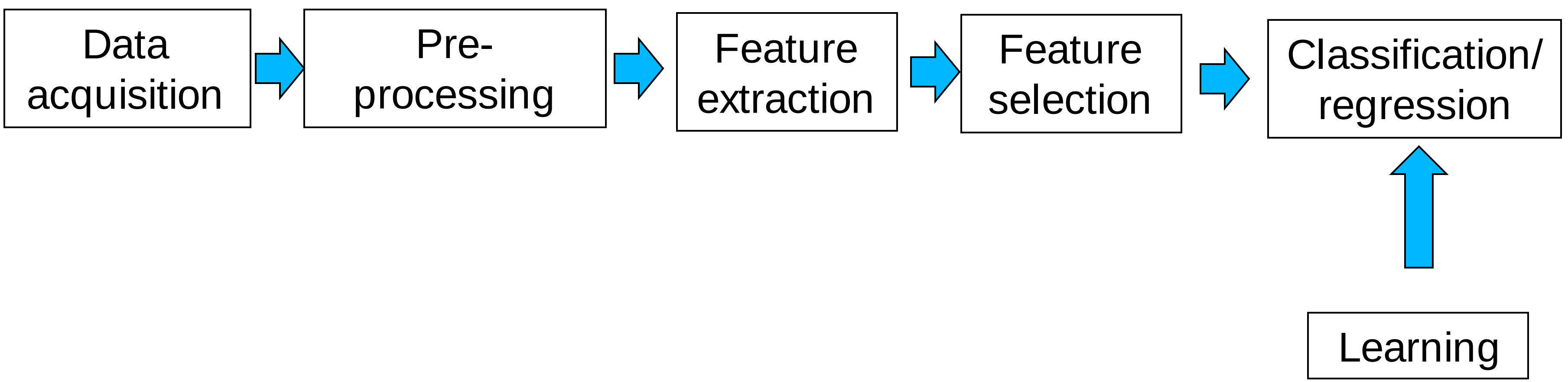
Milestones in the Development of Neural Networks



https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

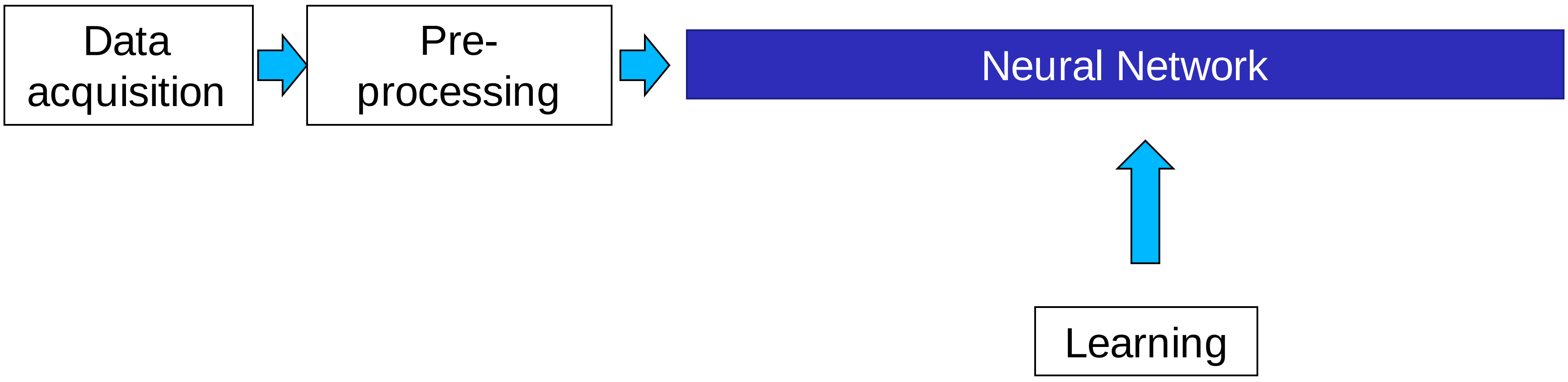
Core Idea: Feature Learning

Classical pattern recognition pipeline



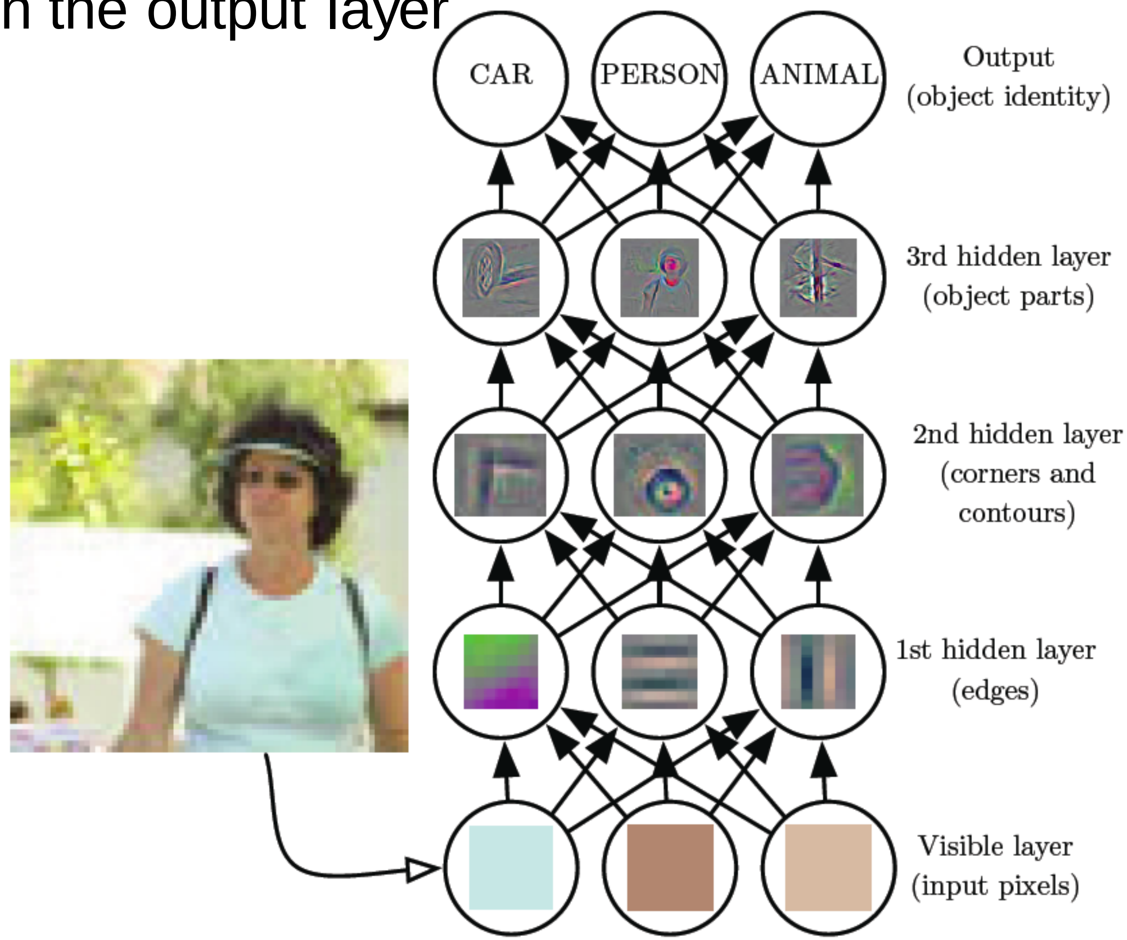
Core Idea: Feature Learning

Neural networks pipeline



Deep Neural Network in action

- Learning representations with increasing level of abstraction
- By passing it with several layers hierarchically, we can classify the images in the output layer



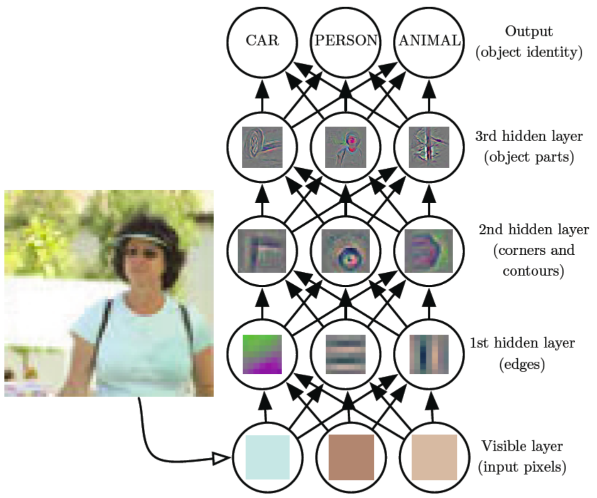
More complex features

Low-level features

Goodfellow, 2016

Deep Neural Network in action

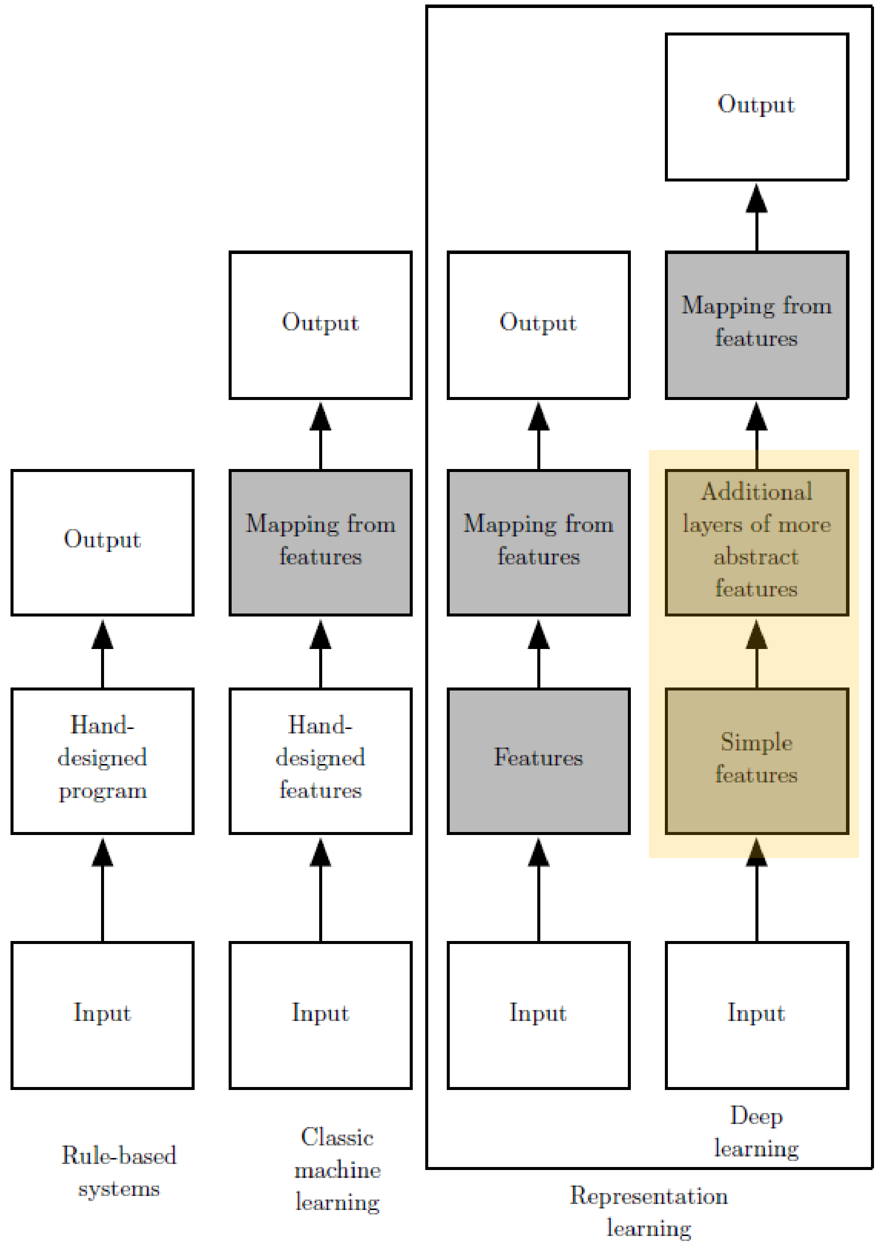
- Image recognition
  - pixel → edge → Texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - sample → spectral band → sound → … → phone → phoneme → word



More complex features

Low-level features

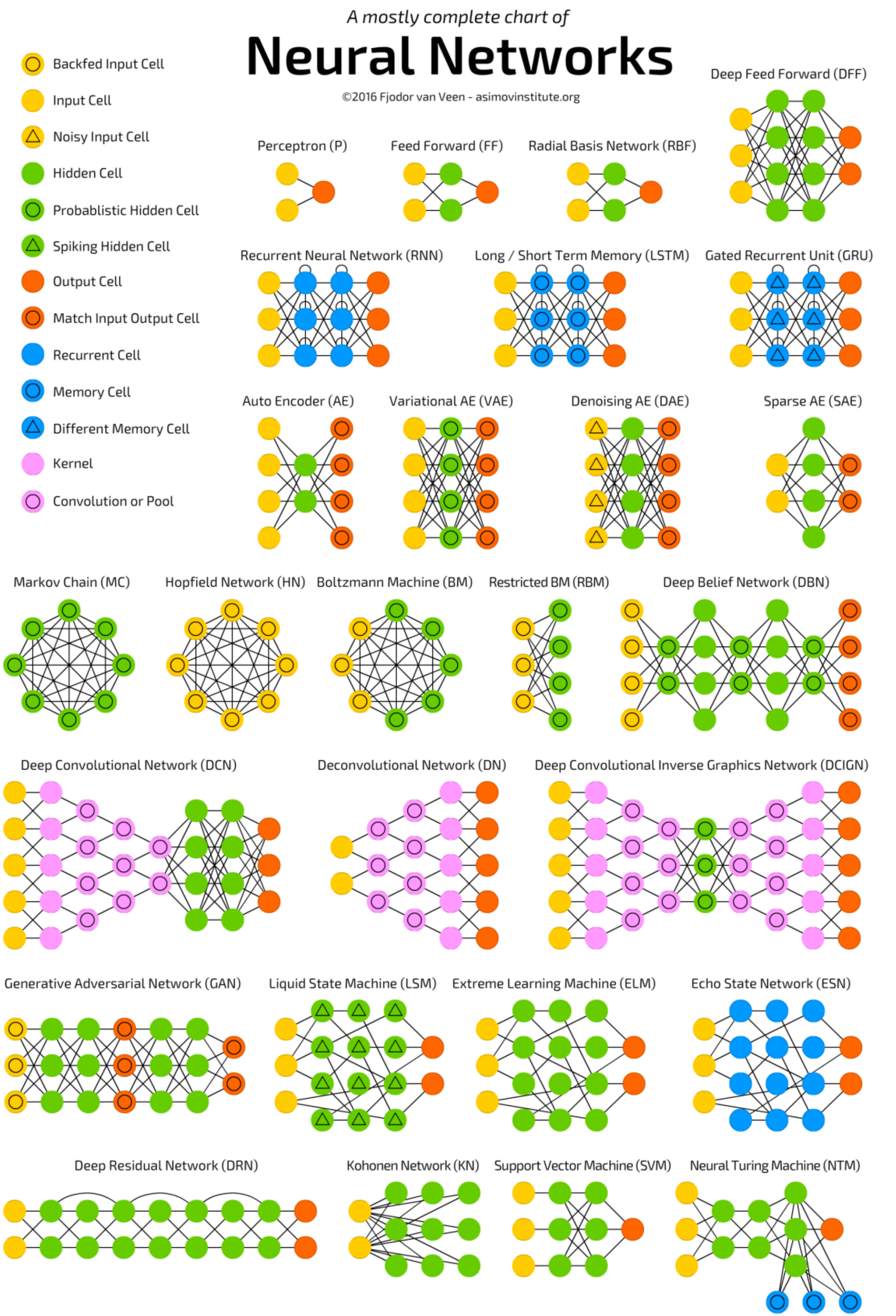
Learning Multiple Components



Automatic feature extraction

Goodfellow, 2016

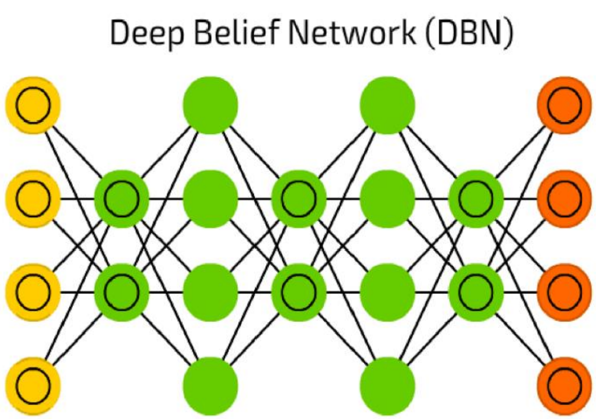
Neural Network Zoo



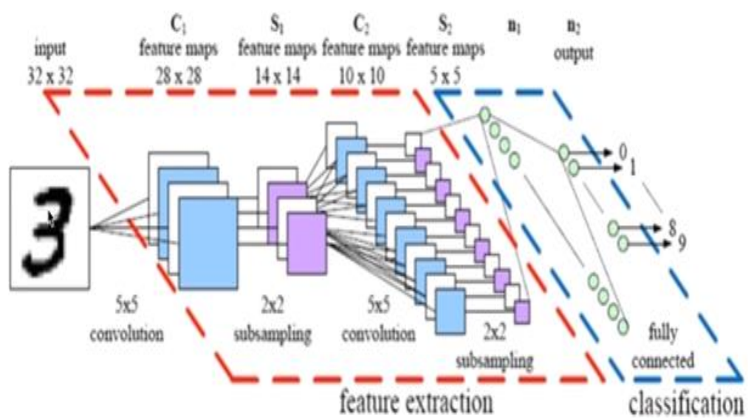
http://www.asimovinstitute.org/neural-network-zoo/

Deep Neural Networks: Applications and Models

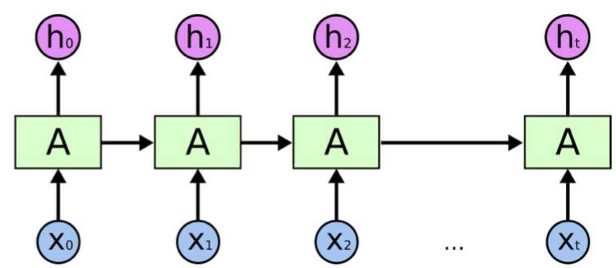
Internet & Cloud	Medicine & Biology	Media & Entertainment	Security & Defense	Autonomous Machines
Image classification Speech recognition Language Translation Language processing Sentiment analysis Recommendation	Cancer cell detection Diabetic grading Drug discovery	Video captioning Video search Real time translation	Face detection Video surveillance Satellite imagery	Pedestrian Detection Lane Tracking Traffic sign recognition



Deep Belief Network

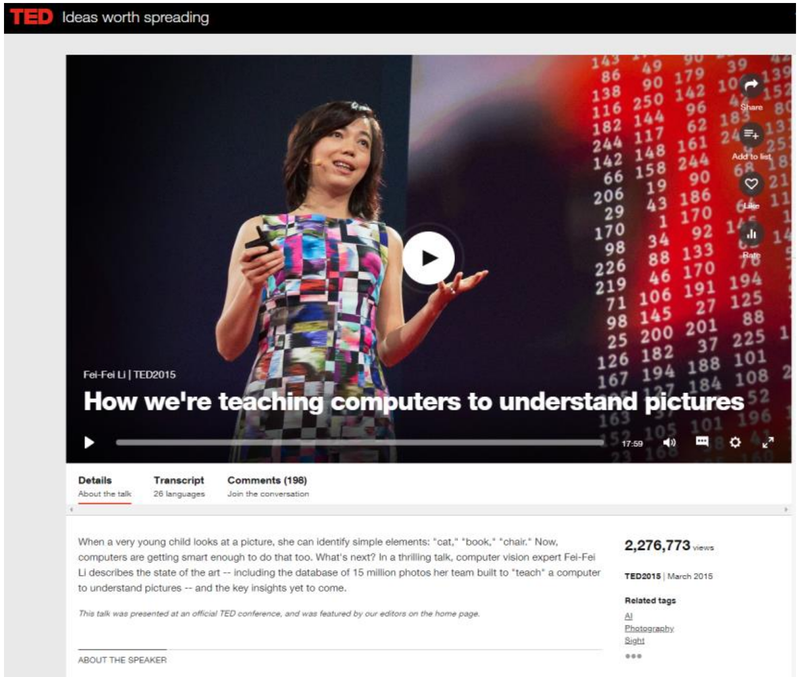


Convolutional Neural Network



Recurrent Neural Network

References on Neural Networks

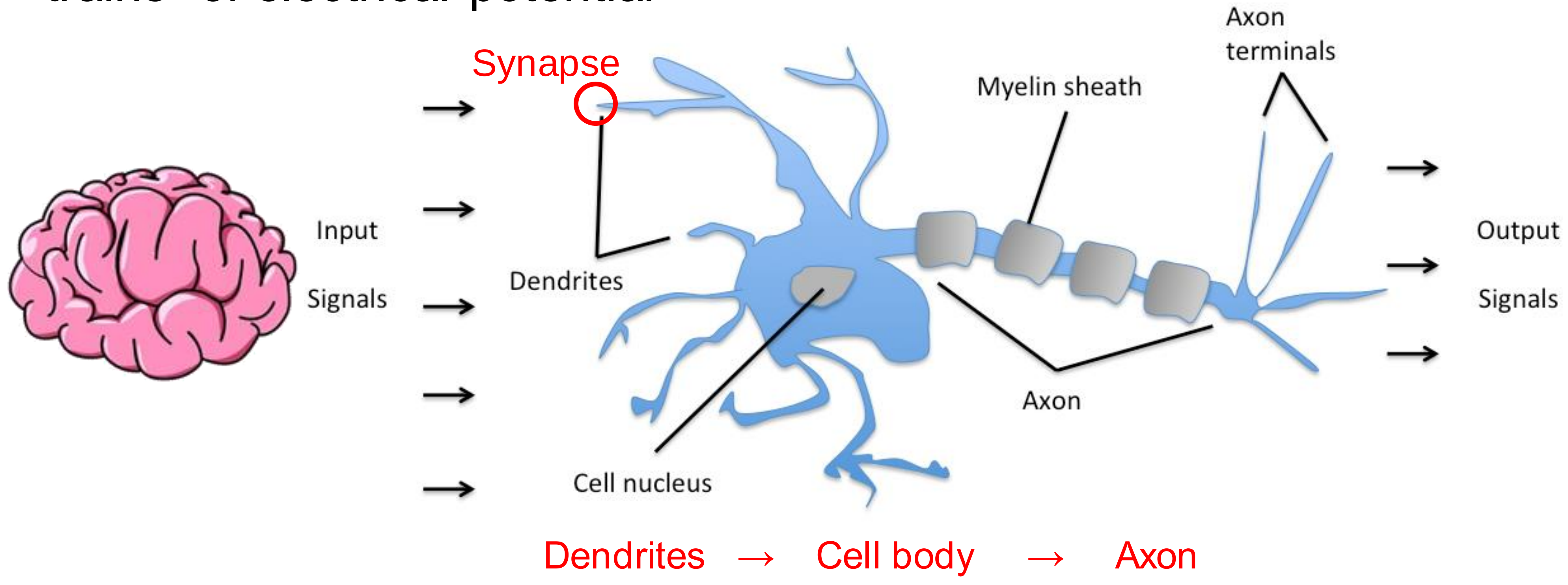


- Andrew Ng's and other ML tutorials
  - https://class.coursera.org/ml-003/lecture
  - http://www.holehouse.org/mlclass/
  - http://deeplearning.stanford.edu/tutorial/
- Stanford lecture videos
  - CS231n: Convolutional Neural Networks for Visual Recognition (http://cs231n.stanford.edu/syllabus.html)
  - CS224d: Deep Learning for Natural Language Processing (http://cs224d.stanford.edu/syllabus.html)
- Ted by Fei-Fei Li (How we're teaching computers to understand pictures (https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures)

Perceptron

Neuronal Activity in the Brain

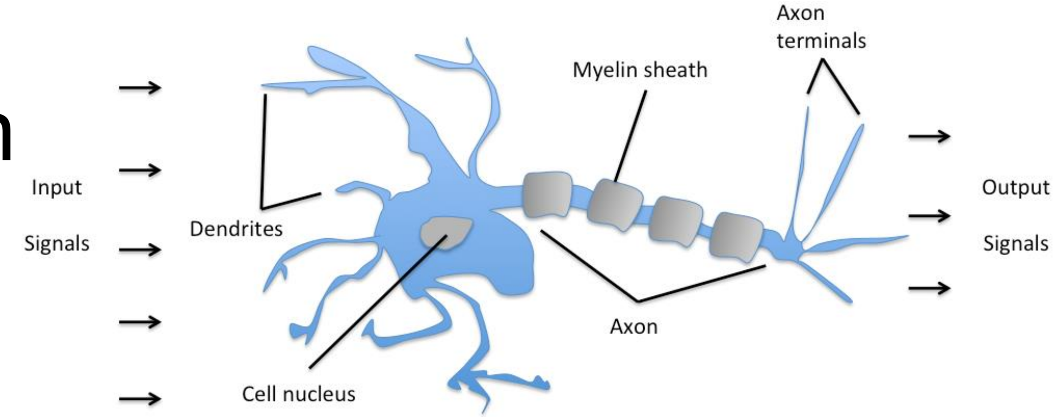
10^{11} neurons of > 20 types, 10^{14} synapses with very complex connections, 1ms–10ms cycle time Signals are noisy "spike trains" of electrical potential



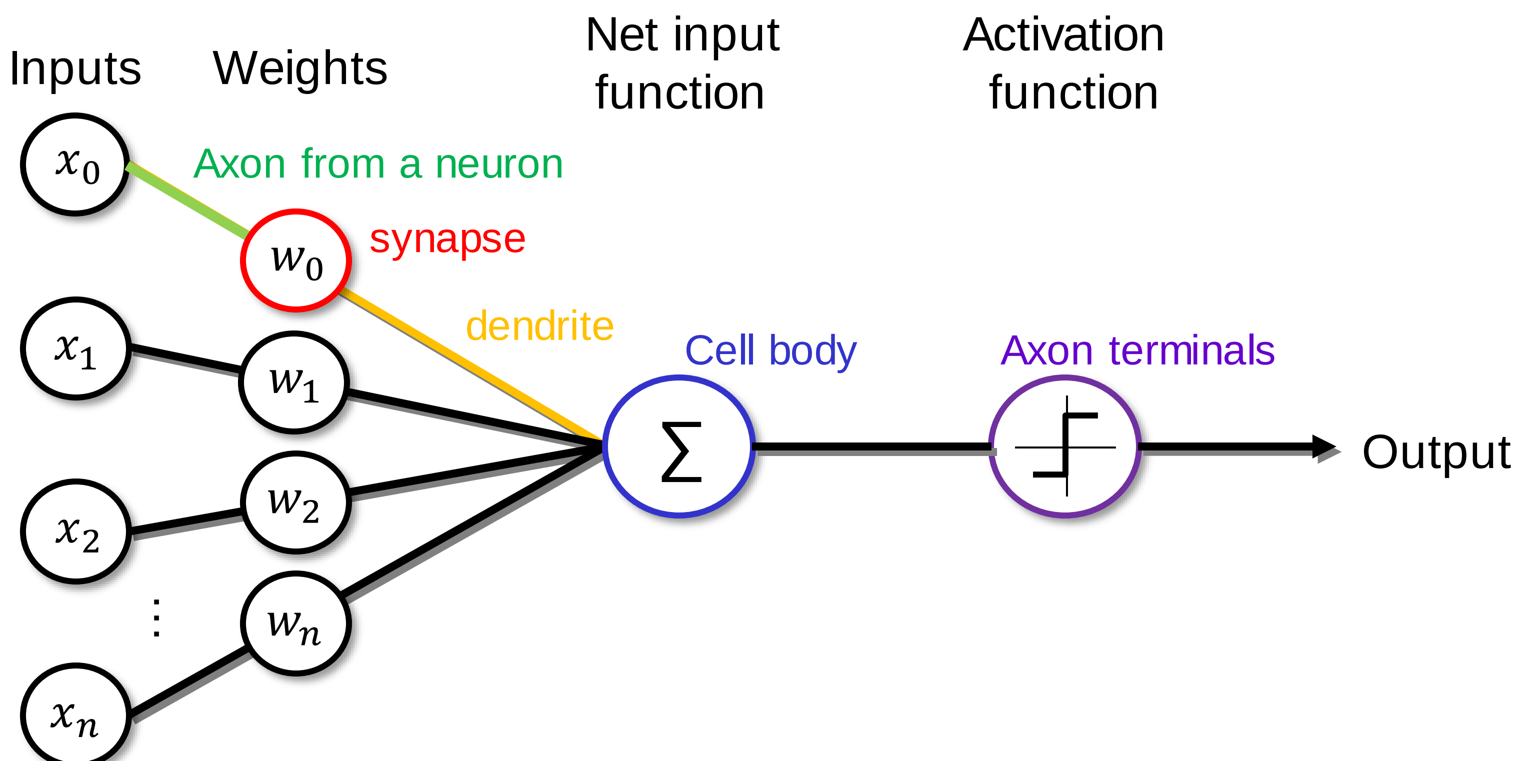
Schematic of a biological neuron.

http://sebastianraschka.com

Rosenblatt's Perceptron

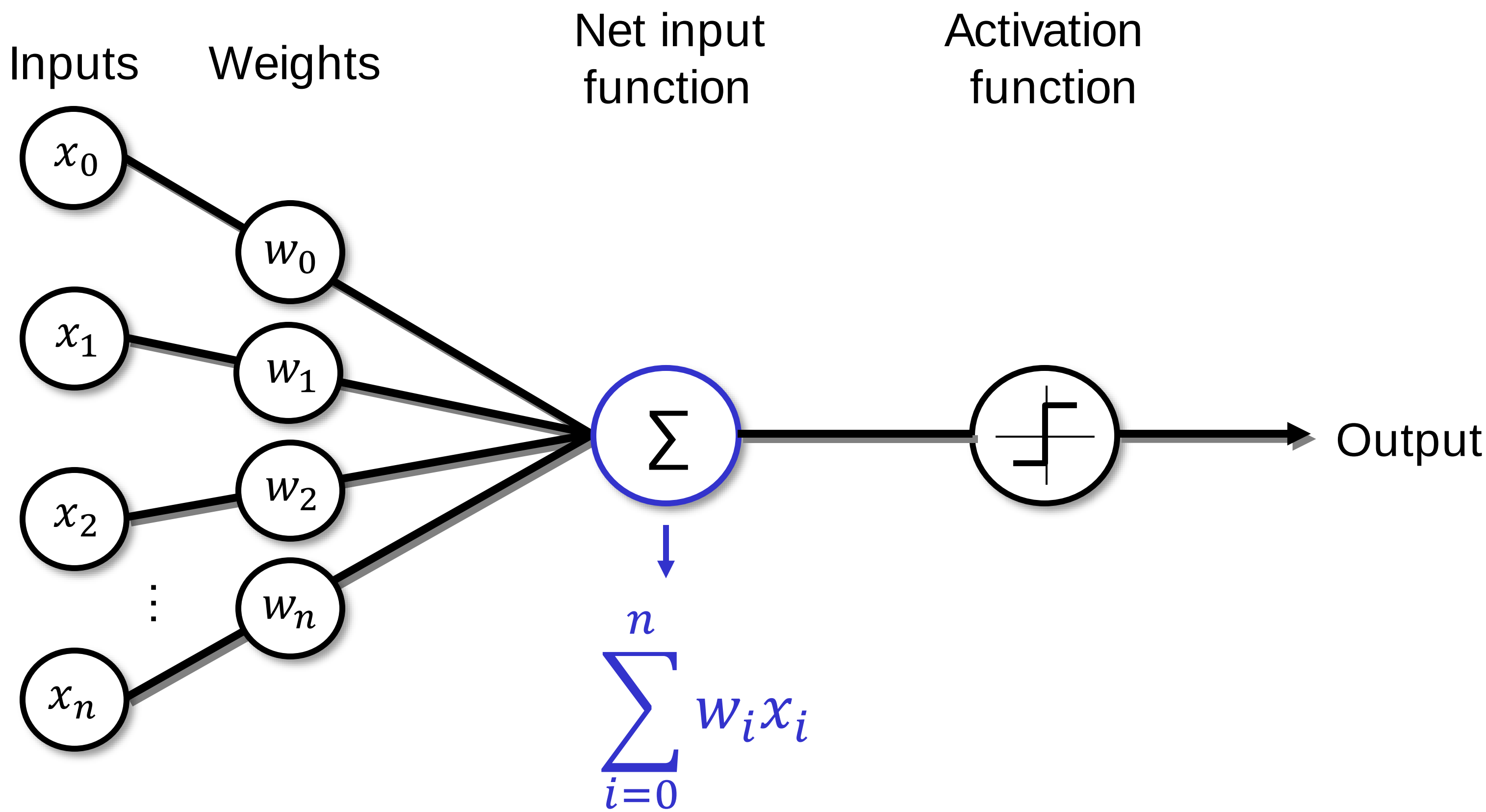


Schematic of a biological neuron.

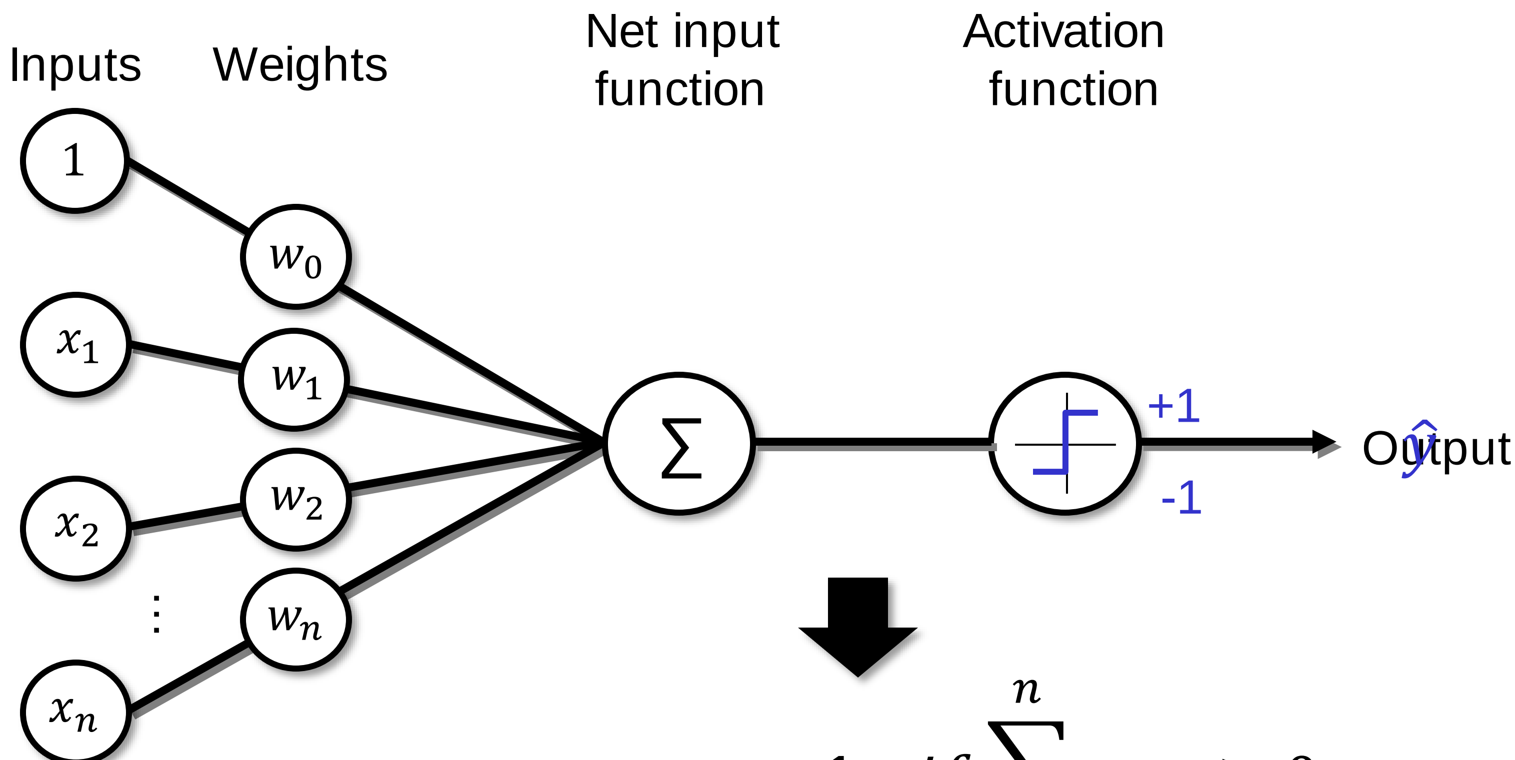


Schematic of Rosenblatt's perceptron

Rosenblatt’s Perceptron: Cell Body



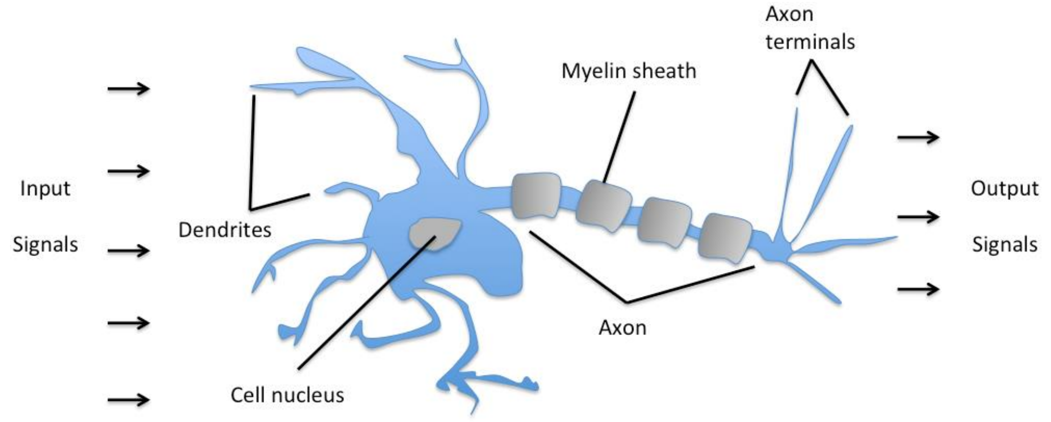
Rosenblatt’s Perceptron: Activation Function



$$\hat{y} = \begin{cases} +1 & if \sum_{i=0}^{n} w_i x_i \geq 0 \\ -1 & otherwise. \end{cases}$$

Perceptron

Nerve cell



Schematic of a biological neuron.

Linear Threshold Unit

(simple) Perceptron
$$\hat{y}(x_1, \ldots, x_n) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron

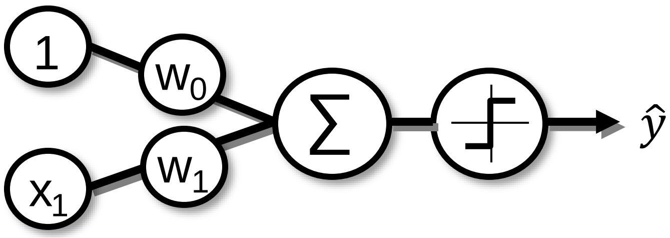
$y = ax + b$ *Linear equation*

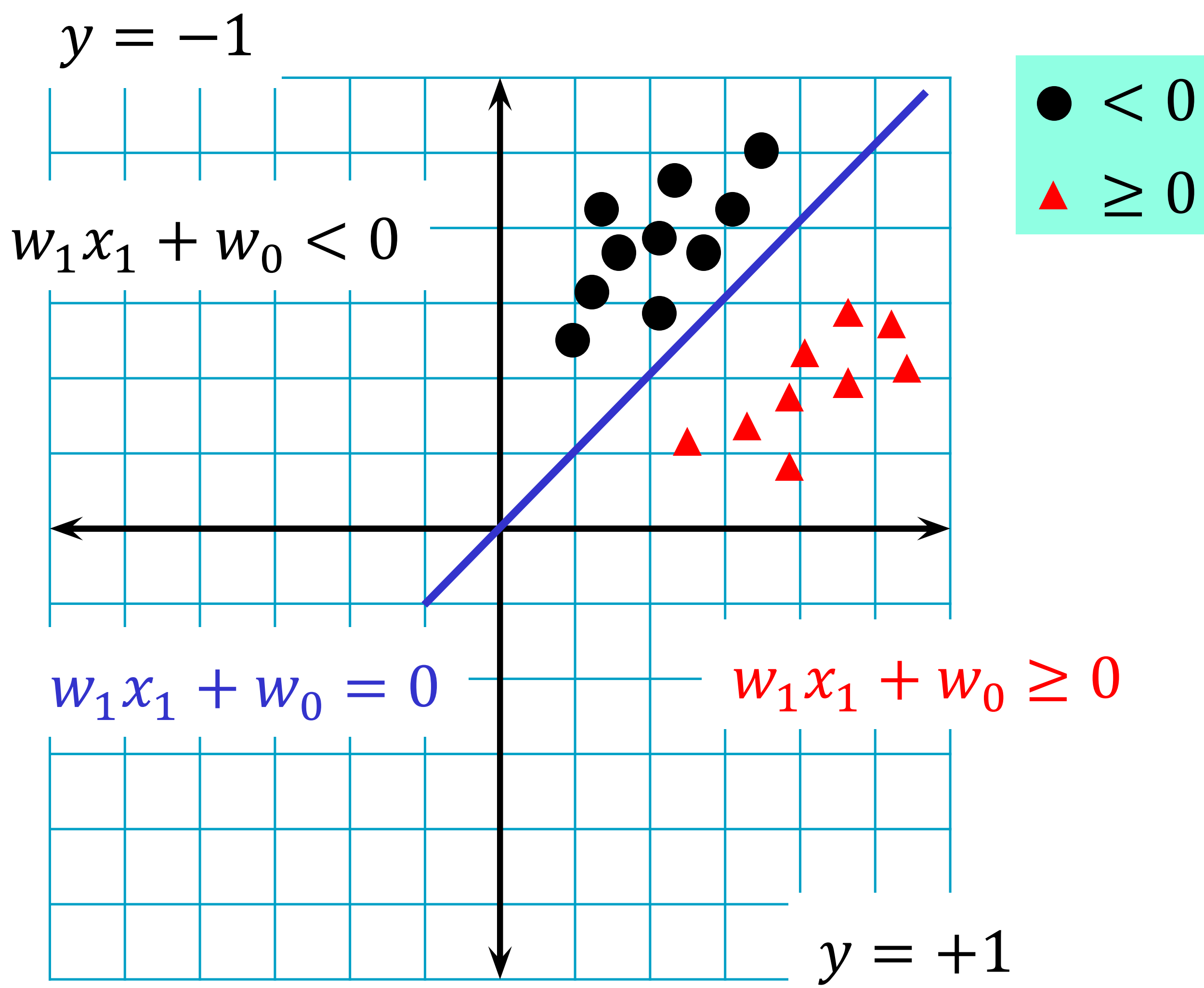
$$\hat{y}(x_1, \ldots, x_n) = \begin{cases} +1 & if\ \boxed{w_0 + w_1 x_1} + \cdots + w_n x_n \geq 0 \\ -1 & otherwise. \end{cases}$$

$$\hat{y}(x_1, \ldots, x_n) = \begin{cases} +1 & if\ \boldsymbol{w}^T \mathbf{x} \geq 0 \\ -1 & otherwise. \end{cases}$$

Perceptron on 2-D coordinate

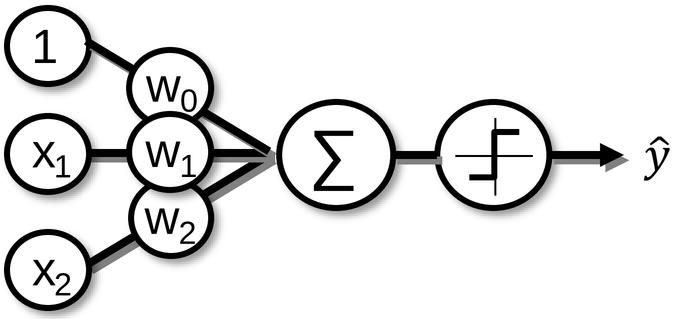
In case if $i = 1$





Perceptron on 2-D coordinate

In case if $i = 2$

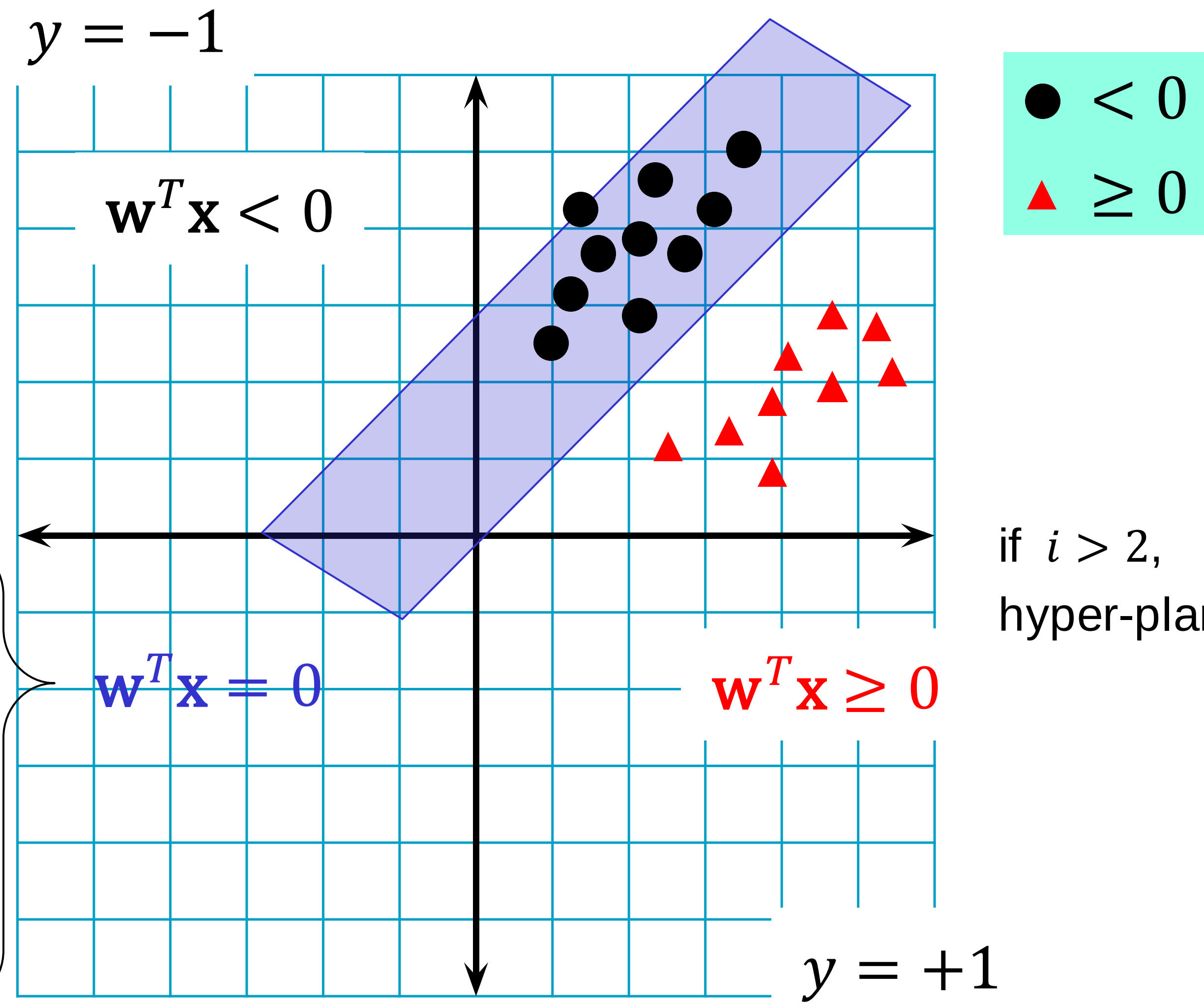


$$w_1 x_1 + w_0 x_0 = 0$$

$$(w_0\ w_1)\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = 0$$

$$\mathbf{w}^T = (w_0\ w_1)$$

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$



if $i > 2$,
hyper-plane

Learning on Perceptron

$$\hat{y} = \mathbf{w}^T\mathbf{x}$$
$$= w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

Delta rule of Rosenblatt's perceptron

1. Initialize the weights
2. For each training sample $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$,
    1. Compute the output $\hat{y}$
    2. Update the weight using Δw_i
3. Repeat 2nd step

Delta Update (=Training) Rule

1. Initialize the weights
2. For each training sample $(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(d)}, y^{(d)}) \in D$, perform the following:
   1. Compute the prediction output $\hat{y} = \mathbf{w}^T \mathbf{x}$
   2. Update the weight $w_i^{new} \leftarrow w_i^{old} + \Delta w_i$

      $(\Delta w_i = -\eta \times \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)} \right) \times x_i)$
3. Repeat until the error is less than threshold

$$\Delta w_i = -\eta \times \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)} \right) \times x_i$$

$y^{(d)}$: Target (correct) output for sample d (0 or 1)
$\hat{y}^{(d)}$: Perceptron output (continuous value)
η: Learning rate (range [0 1])

Perceptron: Cost(= Loss) Function

Loss: $E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d\in D}\underbrace{\left(y^{(d)} - \hat{y}^{(d)}\right)^2}$

$y^{(d)}$: target (0 or 1)
$\hat{y}^{(d)}$: output
$\hat{y} = w_0 + w_1 x_1 + \cdots + w_1 x_1$

Difference between target value $y^{(d)}$ and output value $\hat{y}^{(d)}$ for training sample d

$$\underset{\mathbf{w}}{\text{minimize}}\, E(\mathbf{w})$$

Our objective is to find $\mathbf{w}$ which can minimize cost function

How cost(W) looks like?

$$E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d\in D}\left(y^{(d)} - \hat{y}^{(d)}\right)^2 \quad \Rightarrow \quad E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d\in D} (y^{(d)} - \mathbf{w}^T\mathbf{x}^{(d)})^2$$

$\mathbf{x}$	y
1	1
2	2
3	3

- $w = 1$, $E(\mathbf{w})$?

How cost(W) looks like?

$$E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d \in D} (y^{(d)} - \mathbf{w}^T \mathbf{x}^{(d)})^2$$

$\mathbf{x}$	y
1	1
2	2
3	3

- $w = 1,\ E(\mathbf{w}) = 0$

  $\frac{1}{2}\left((1 - 1 \times 1)^2 + (2 - 1 \times 2)^2 + (3 - 1 \times 3)^2\right)$

How cost(W) looks like?

$$E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d \in D}(y^{(d)} - \mathbf{w}^T\mathbf{x}^{(d)})^2$$

$\mathbf{x}$	y
1	1
2	2
3	3

- $w = 1,\ E(\mathbf{w}) = 0$

  $\frac{1}{2}\left((1 - 1 \times 1)^2 + (2 - 1 \times 2)^2 + (3 - 1 \times 3)^2\right)$

- $w = 0,\ E(\mathbf{w}) = ?$

- $w = 2,\ E(\mathbf{w}) = ?$

How cost(W) looks like?

$$E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d \in D} (y^{(d)} - \mathbf{w}^T \mathbf{x}^{(d)})^2$$

$\mathbf{x}$	y
1	1
2	2
3	3

- $w = 1,\ E(\mathbf{w}) = 0$

  $\frac{1}{2}\left((1 - 1 \times 1)^2 + (2 - 1 \times 2)^2 + (3 - 1 \times 3)^2\right)$

- $w = 0,\ E(\mathbf{w}) = 4.5$

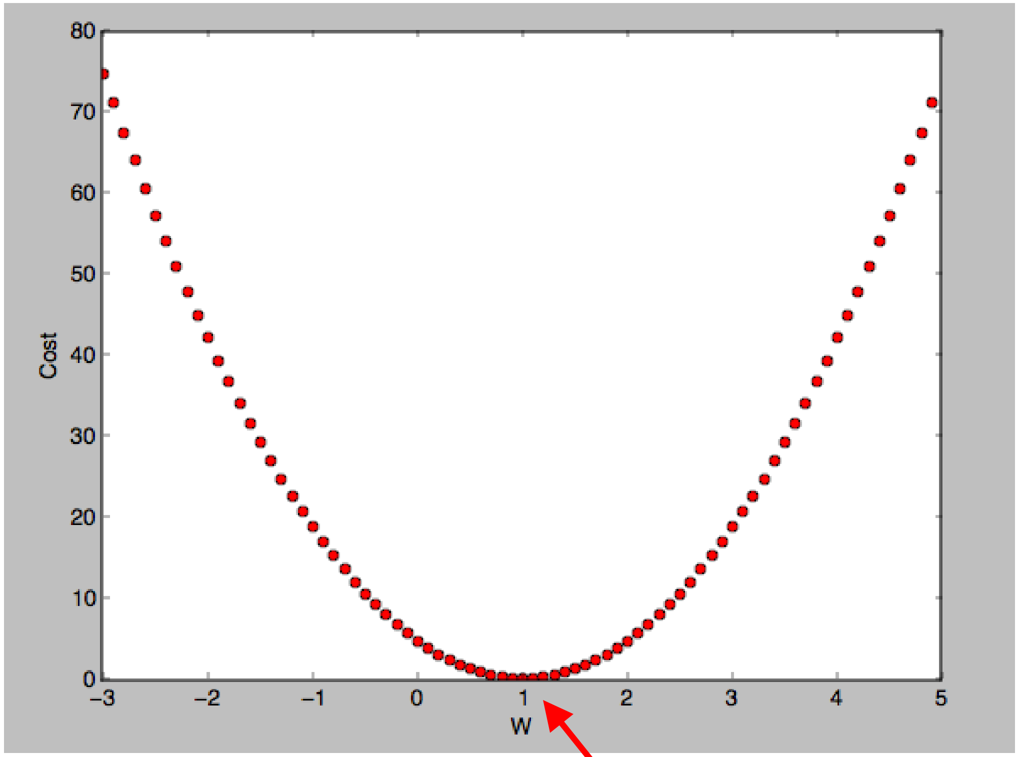
  $\frac{1}{2}\left((1 - 0 \times 1)^2 + (2 - 0 \times 2)^2 + (3 - 0 \times 3)^2\right)$

- $w = 2,\ E(\mathbf{w}) = 4.5$

  $\frac{1}{2}\left((1 - 2 \times 1)^2 + (2 - 2 \times 2)^2 + (3 - 2 \times 3)^2\right)$

How cost(W) looks like?

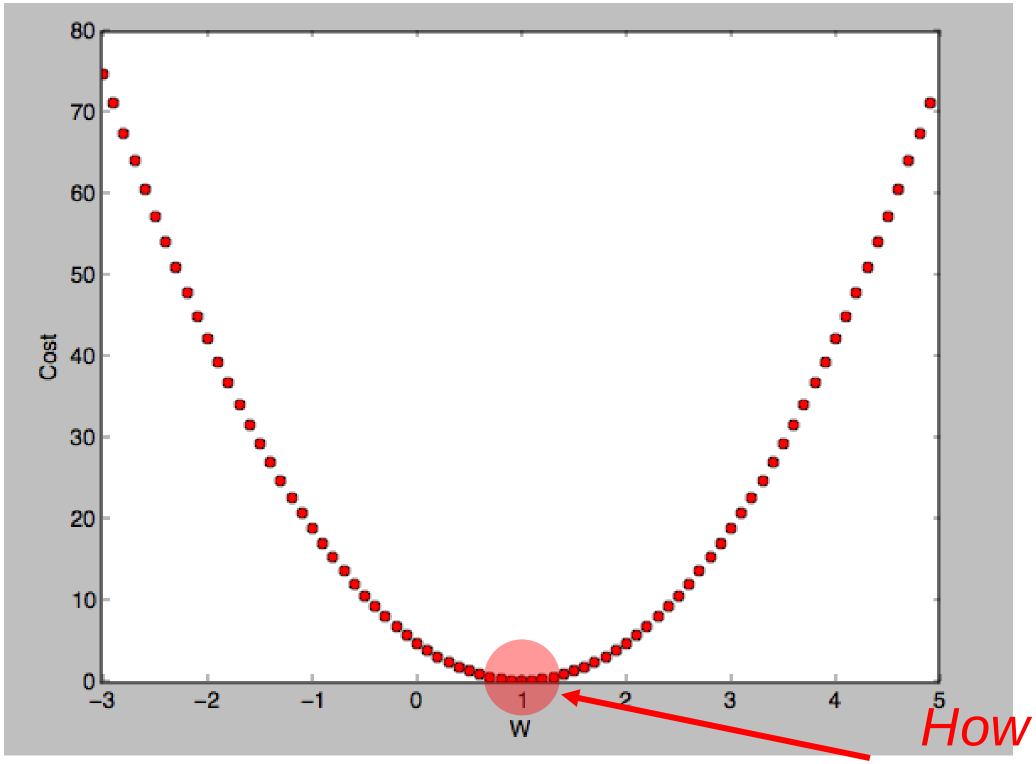
$$E(\mathbf{w}) \equiv \frac{1}{2}\sum_{d\in D}(y^{(d)} - \hat{y}^{(d)})^2$$

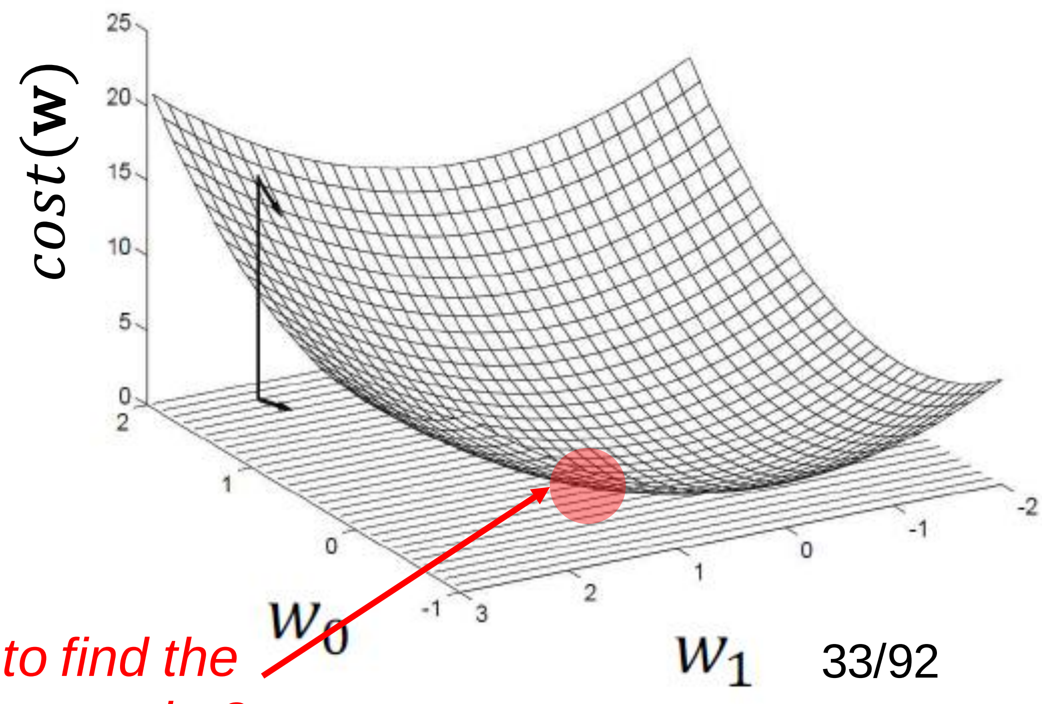


Minimum point = 1

How to minimize cost?

$$E(\mathbf{w}) \equiv \frac{1}{2} \sum_{d \in D} (y^{(d)} - \hat{y}^{(d)})^2$$

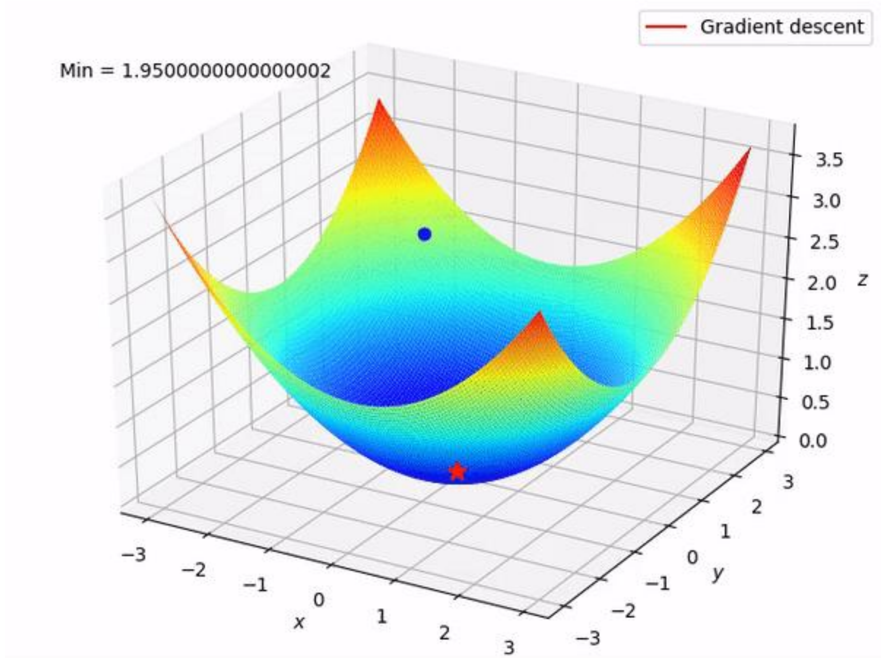




How to find the minimum point?

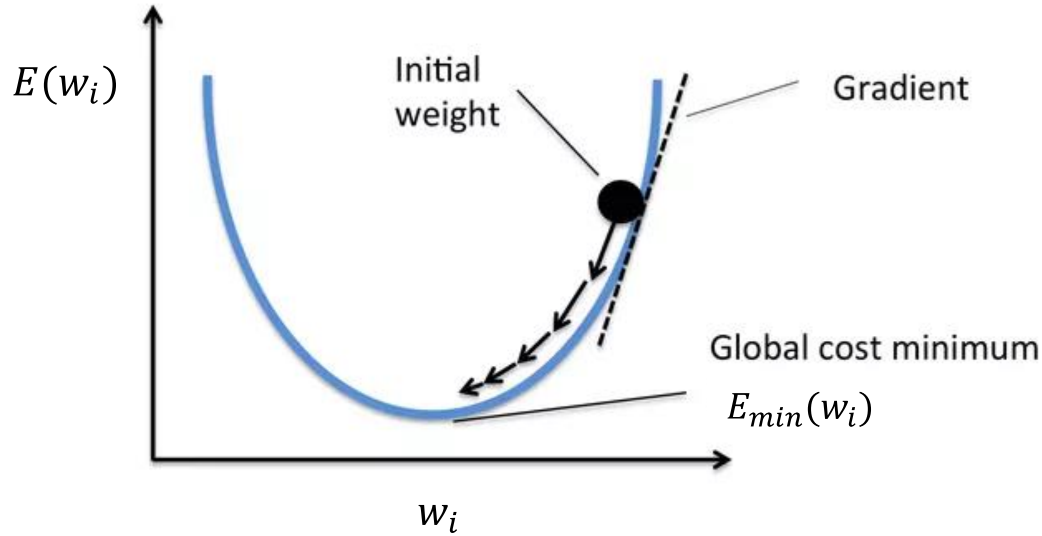
Gradient Descent Algorithm

- Minimize cost function
- Gradient descent is used for many minimization problems
- For a given cost function, $E(\mathbf{w})$, it will find w to minimize cost
- Repeat until you converge to a local minimum



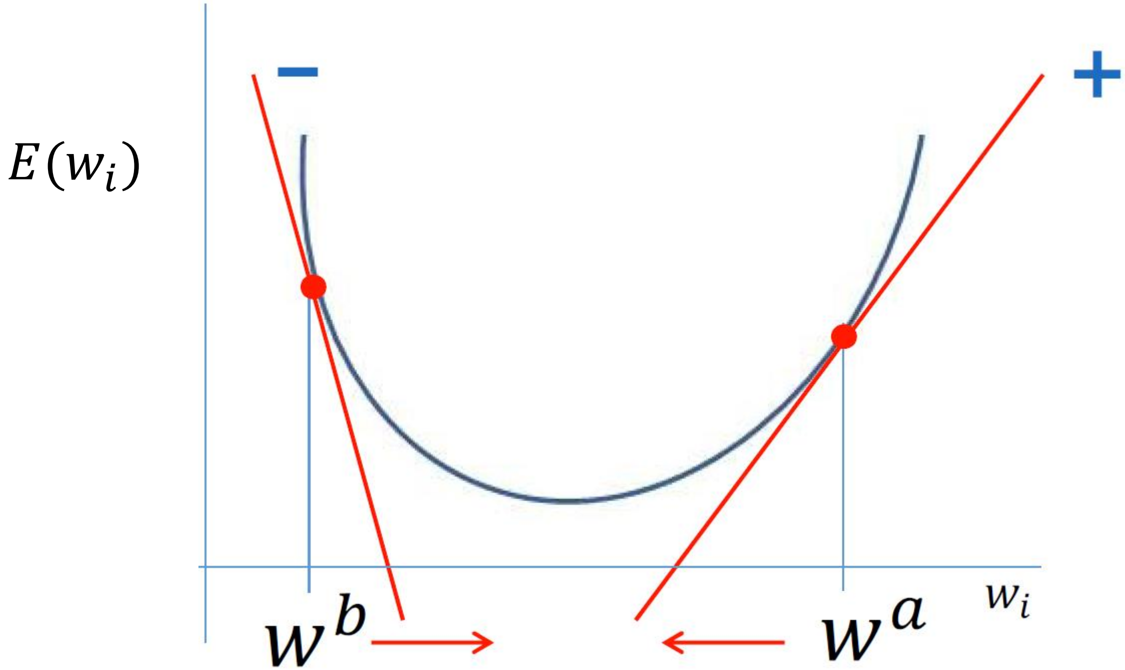
Gradient Descent Algorithm

How it works?



1. Start with initial guesses
   - Start at random value
2. Each weight is updated by taking a step into the opposite direction of the gradient $\Delta w_i = -\eta \times \frac{\partial E}{\partial w_i}$
   - Compute the partial derivative of the cost function $\frac{\partial E}{\partial w_i}$ for each weight
3. Repeat until you converge to a local minimum

Gradient Descent Algorithm



$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

η: learning rate (*e.g.* 0.001)

$\frac{\partial E}{\partial w_i}$: gradient

Gradient descent algorithm

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d \left(y^{(d)} - \hat{y}^{(d)}\right)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}\left(y^{(d)} - \hat{y}^{(d)}\right)^2$$

$$= \frac{1}{2}\sum_d 2\left(y^{(d)} - \hat{y}^{(d)}\right)\frac{\partial}{\partial w_i}\left(y^{(d)} - \hat{y}^{(d)}\right)$$

$$= \sum_d \left(y^{(d)} - \hat{y}^{(d)}\right)\frac{\partial}{\partial w_i}\left(y^{(d)} - w_0 x_0 - w_1 x_1 - \cdots - w_i x_i\right)$$

$$= \sum_d \left(y^{(d)} - \hat{y}^{(d)}\right) \times (-x_i)$$

$$\Delta w_i = \eta \times \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)}\right) \times (-x_i)$$

Delta Rule based on Gradient Descent Algorithm (Summary)

$$\hat{y} = w_0 1 + w_1 x_1 + \cdots + w_n x_n$$

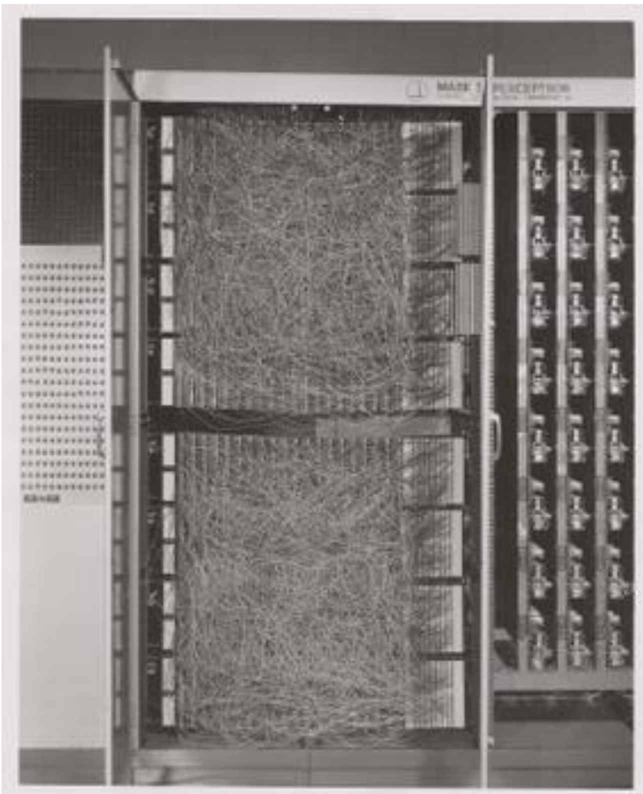
$$\boxed{E(\mathbf{w}) \equiv \frac{1}{2} \sum_{d} \left(y^{(d)} - \hat{y}^{(d)}\right)^2} \rightarrow$$

Squared loss function

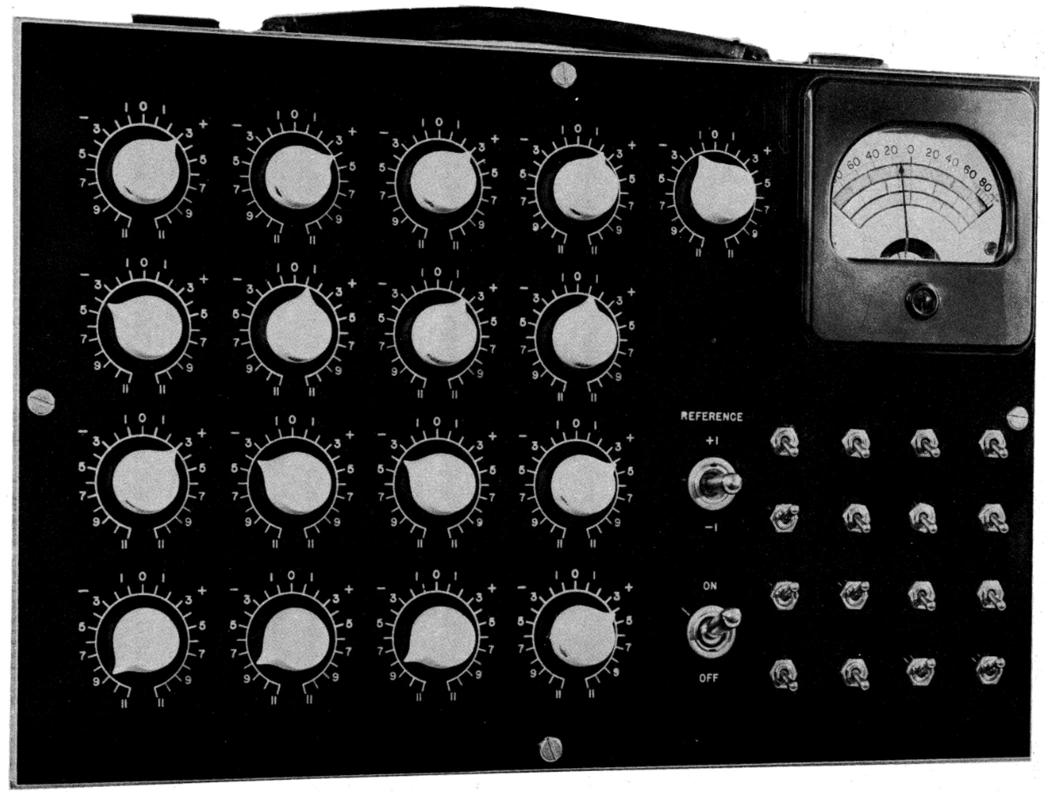
$$w_i^{new} \leftarrow w_i^{old} + \Delta w_i$$

$$\Delta w_i = -\eta \times \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)}\right) \times x_i$$

Hardware implementations



Frank Rosenblatt, ~1957: Perceptron



Widrow and Hoff, ~1960: Adaline/Madaline

False Promises

The New York Times

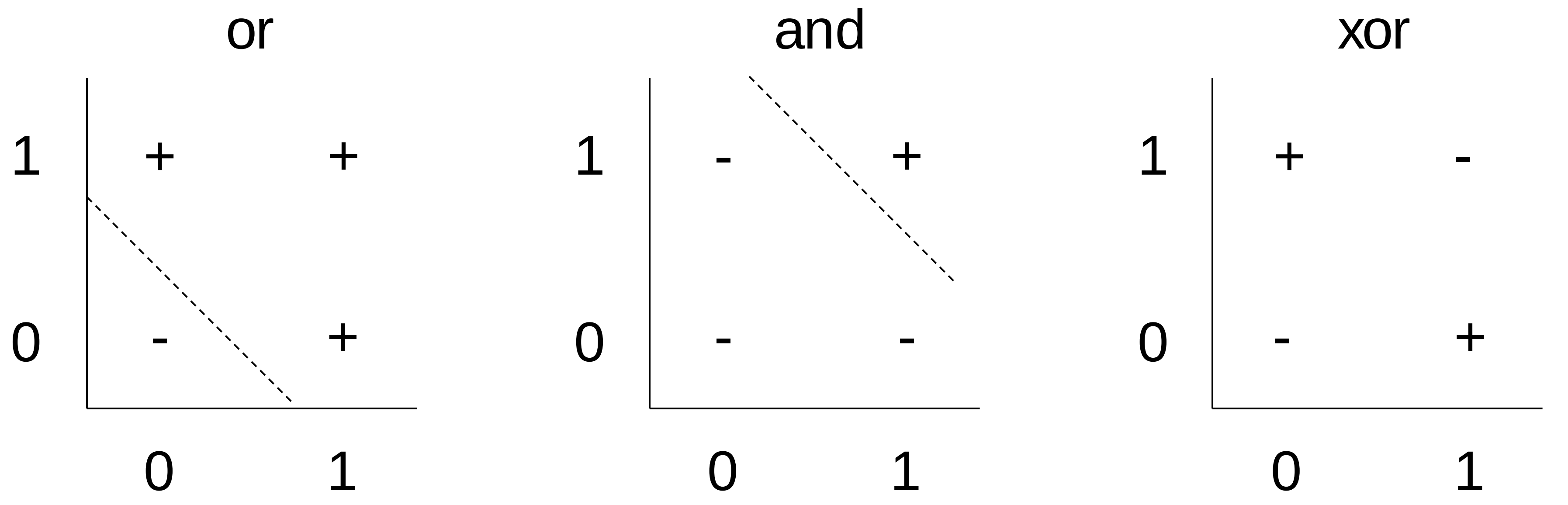
NEW NAVY DEVICE LEARNS BY DOING

July 8, 1958

"The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence… Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers"

Reference: 'NewYork Times' newspaper on July 8, 1958

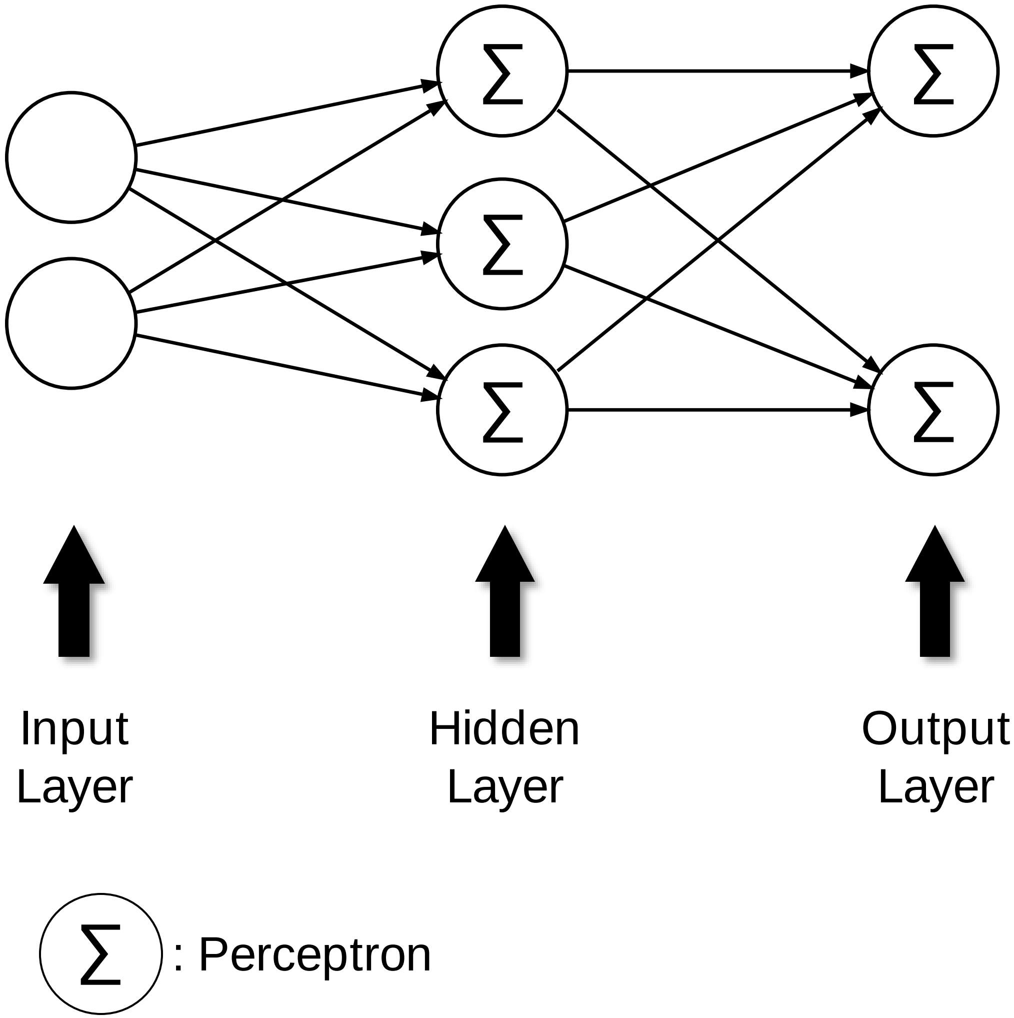
(Simple) AND/OR problem: linearly separable?



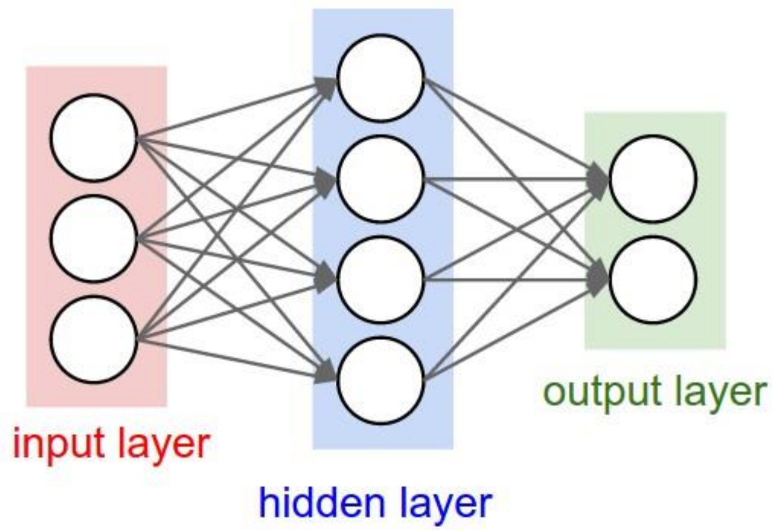
Yep Yep Nope

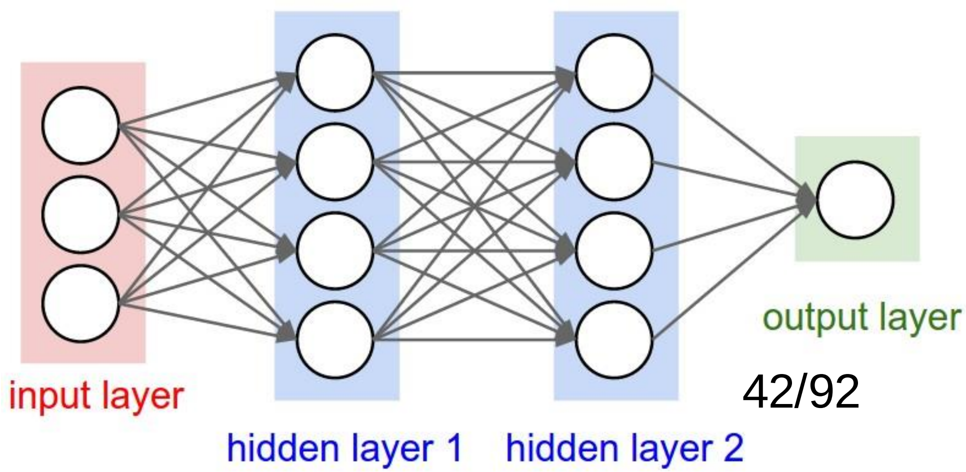
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Multi-Layer Perceptron (by M. Minsky)

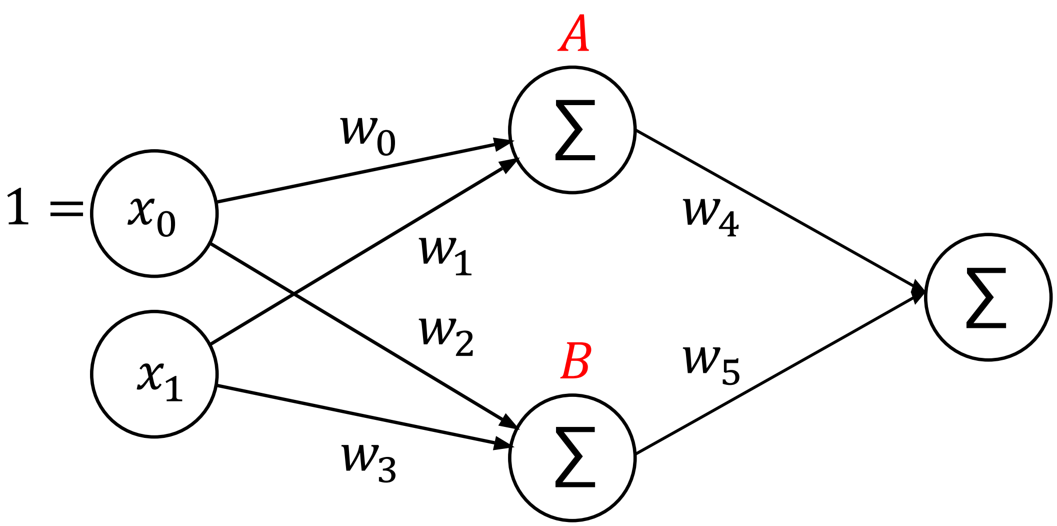


2-Layer Neural Network





Multi-Layer Perceptron: Limitation



$$\hat{y} = w_4 \times \overbrace{(w_0x_0 + w_1x_1)}^{A} + w_5 \times \overbrace{(w_2x_1 + w_3x_1)}^{B}$$

$$= w_4w_0x_0 + w_4w_1x_1 + w_5w_2x_1 + w_5w_3x_1$$

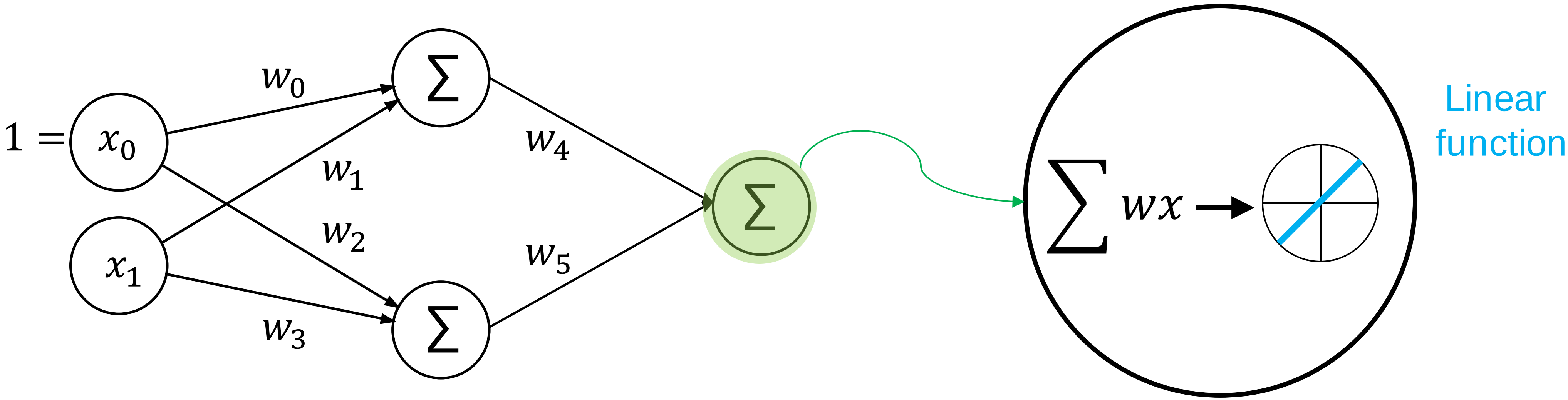
$$= (w_4w_0 + w_5w_2)x_0 + (w_4w_1 + w_5w_3)x_1$$

$$= \underline{(w_4w_0 + w_5w_2)} + \underline{(w_4w_1 + w_5w_3)}x_1$$

$$\hat{y} = w_a + w_b x_1$$

Still Linear equation
(Line, plane, or hyper-plane)

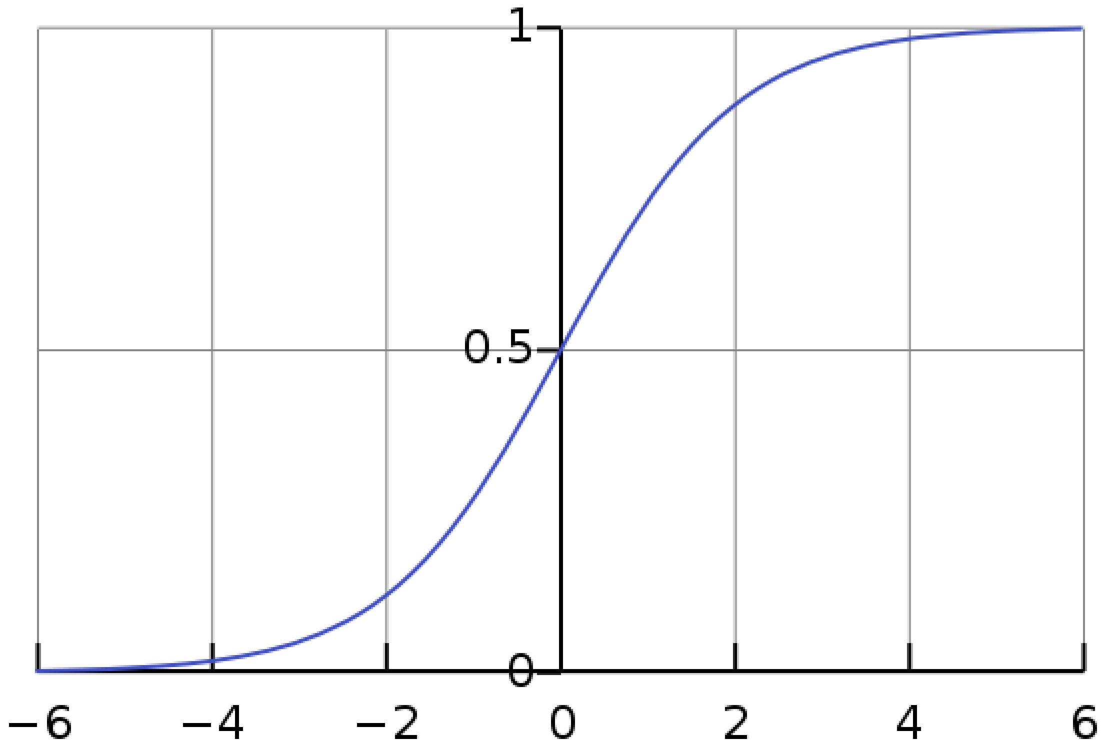
Multi-Layer Perceptron: Limitation



$\hat{y} = f(z) = z$

Multi-Layer Perceptron: Activation Function

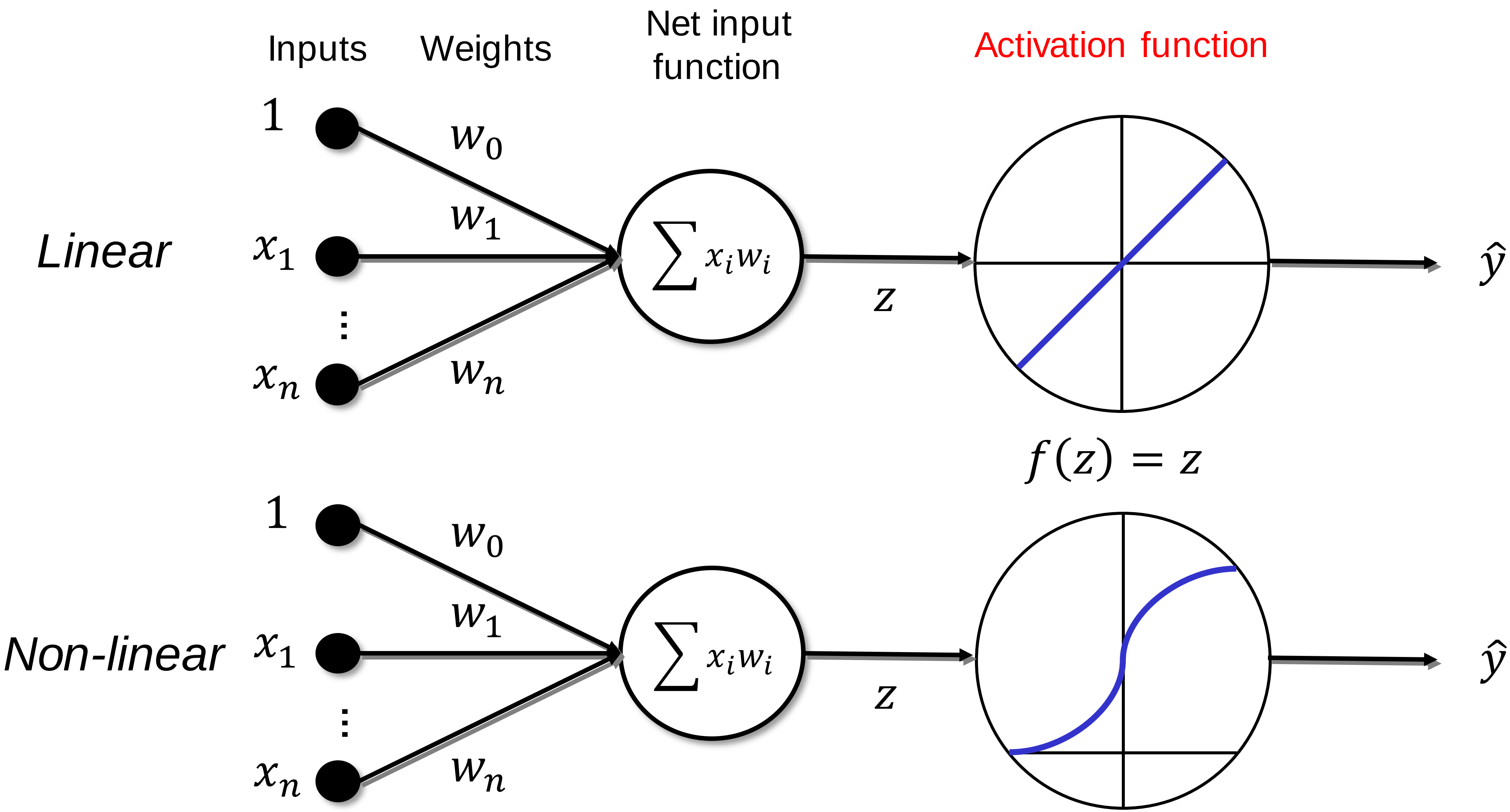
Non-linear function



Sigmoid function

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

Activation Functions



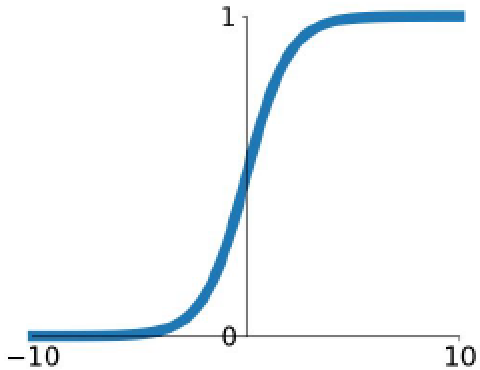
Note: $\frac{d}{dz} f(z) = f(z)(1 - f(z))$

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

Activation Functions

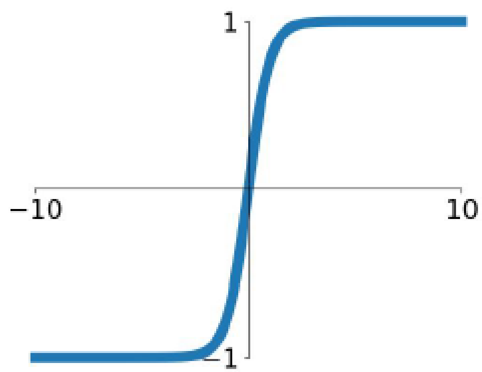
Sigmoid

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



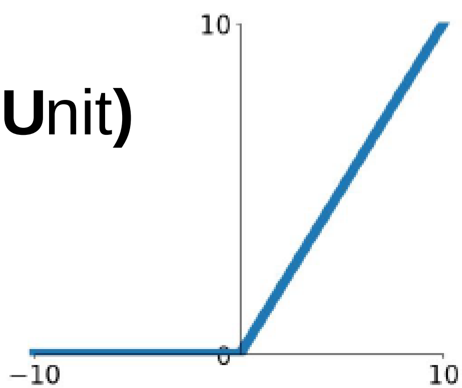
tanh

$$\tanh(x)$$



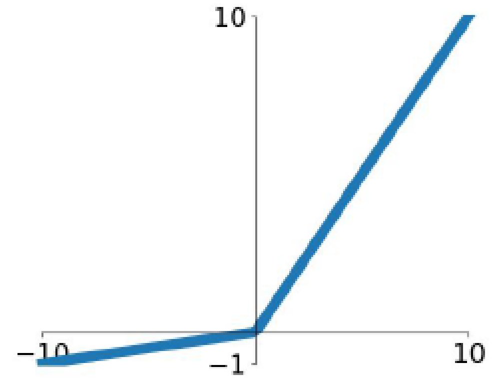
ReLU
(Rectified Linear Unit)

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

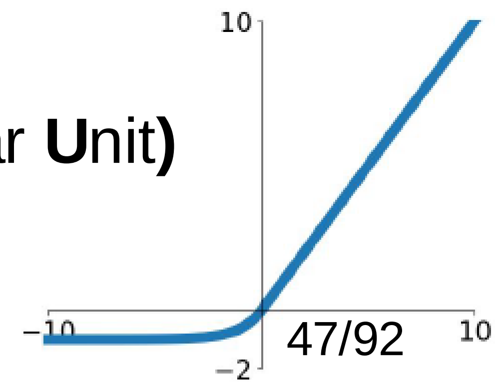


Maxout

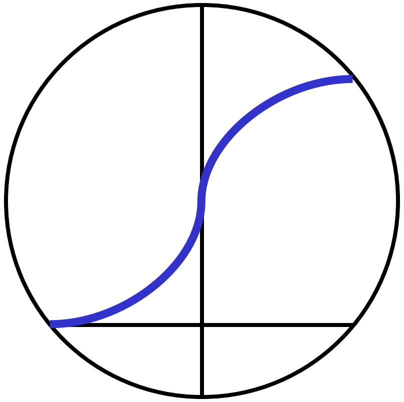
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU
(Exponential Linear Unit)

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Derivative of Sigmoid Function



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

$$\frac{d}{dx} f(x) = f(x)(1 - f(x))$$

$$\frac{d}{dx} f(x) = \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$\left(\frac{d}{dx} e^{x} = e^{x}\right) \qquad = \frac{1-1+e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

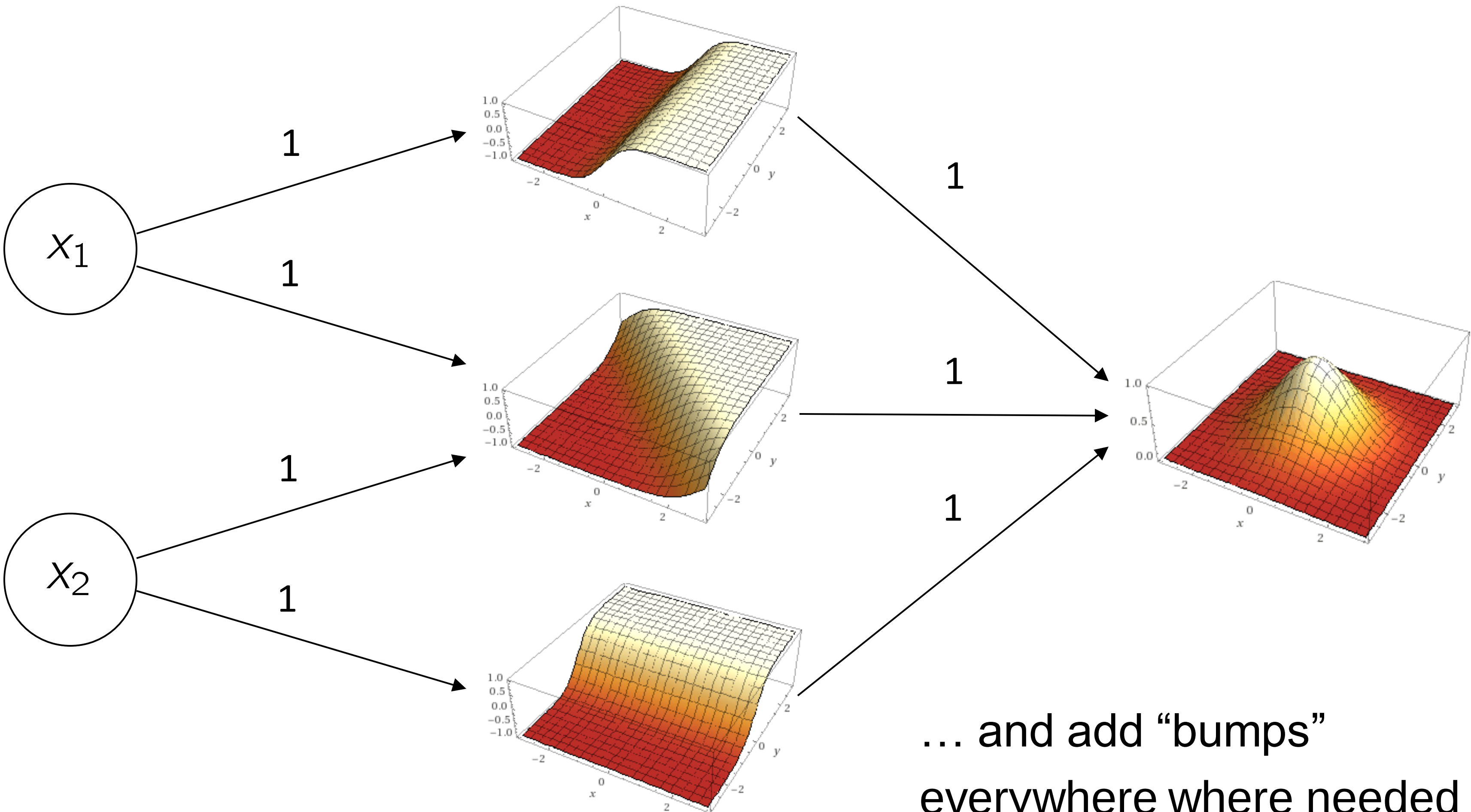
$$= \left(\frac{1}{1+e^{-x}}\right) - \left(\frac{1}{1+e^{-x}}\right)^2 = \left(\frac{1}{1+e^{-x}}\right)\left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= f(x)(1 - f(x))$$

Link: http://www.derivative-calculator.net/

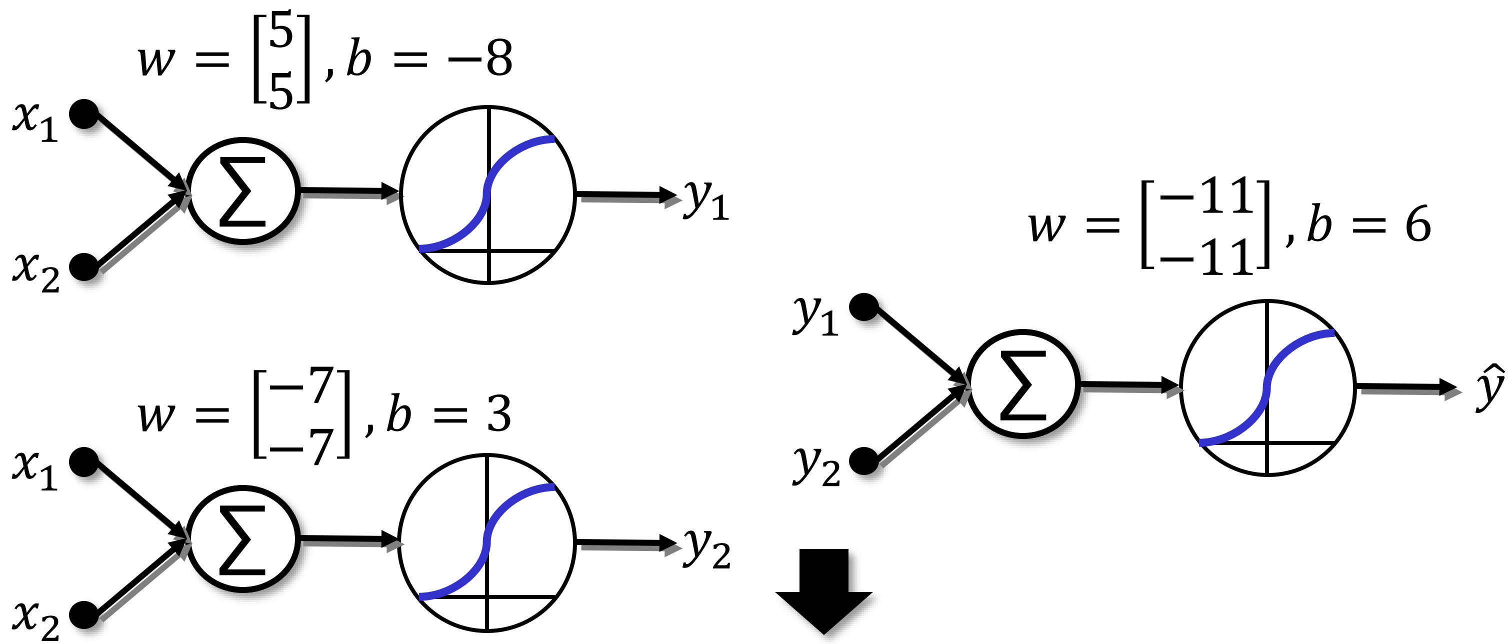
Multi-layer Perceptron is Universal Approximator

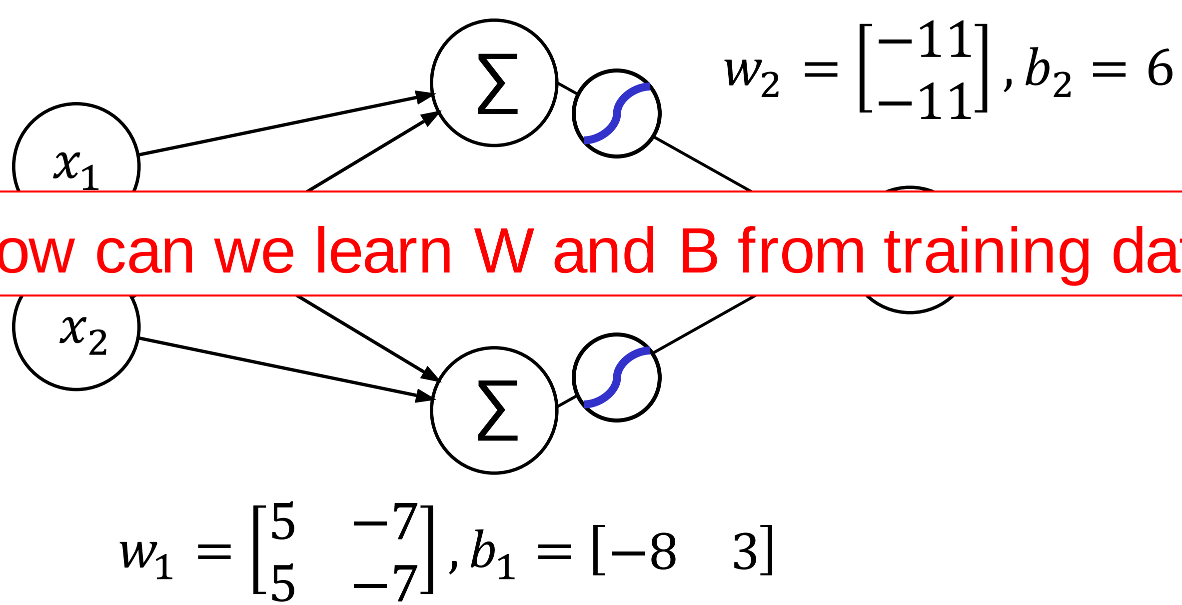
"Proof" by construction:



… and add "bumps" everywhere where needed in the input domain.

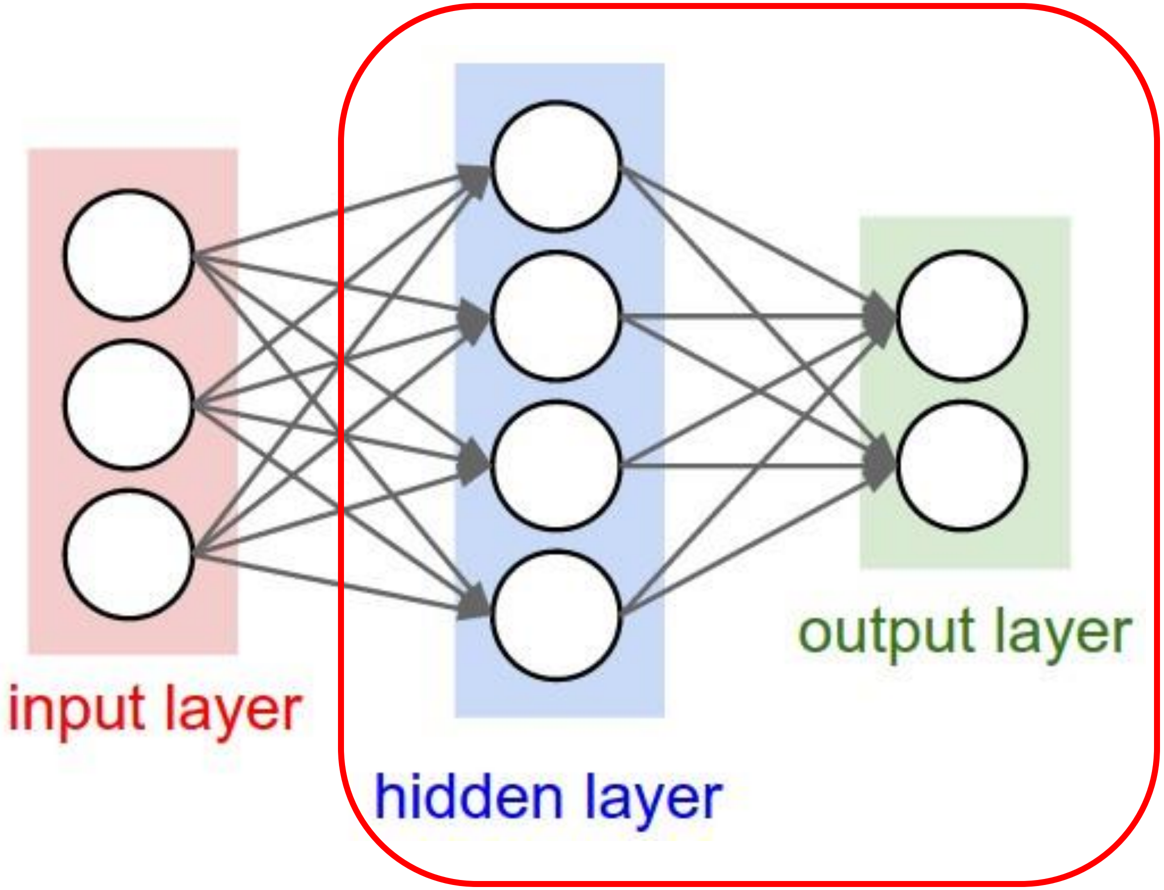
Multi-Layer Perceptron: Forward Propagation





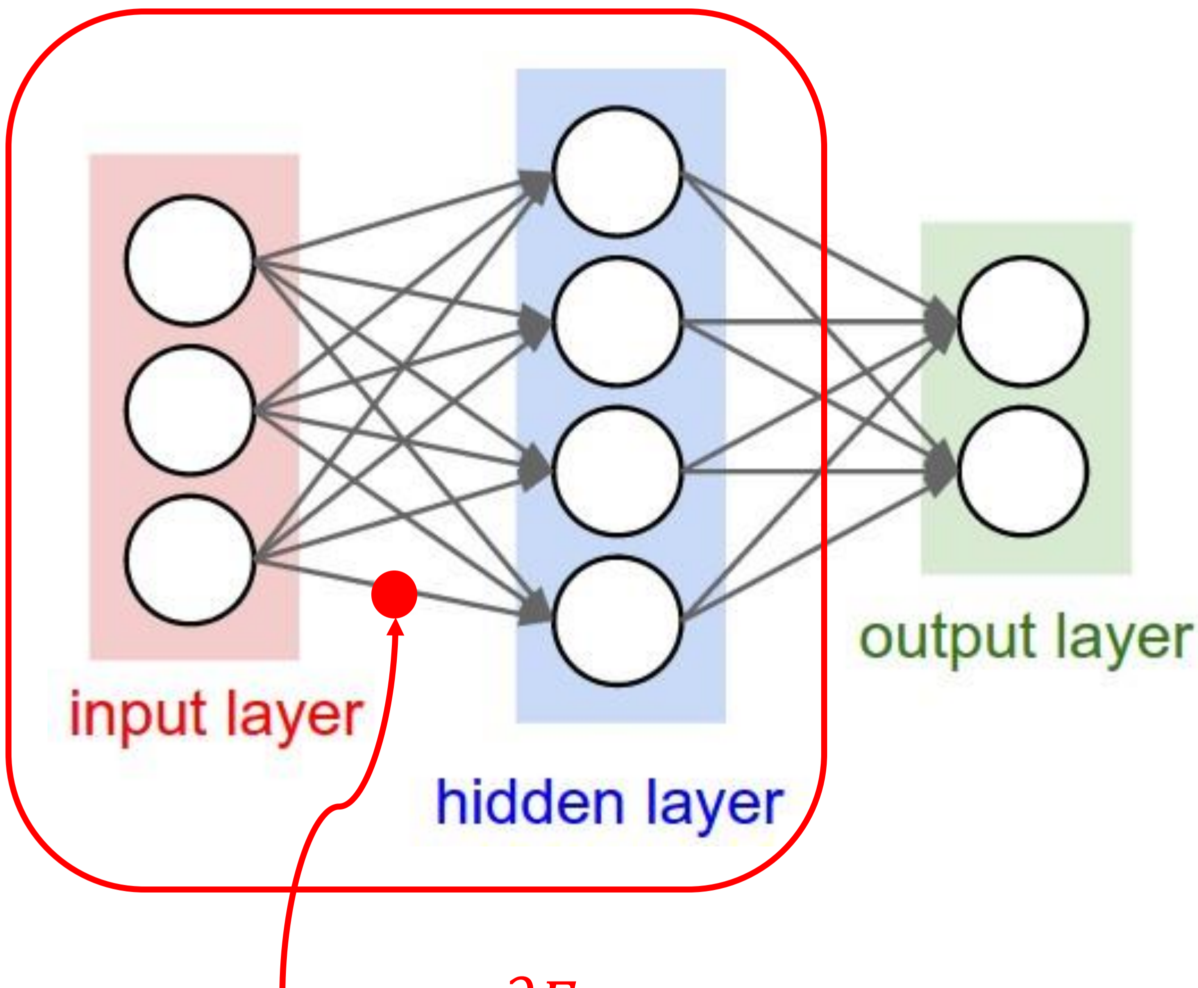
Question: How can we learn W and B from training data?

Learning on Multi-Layer Perceptron



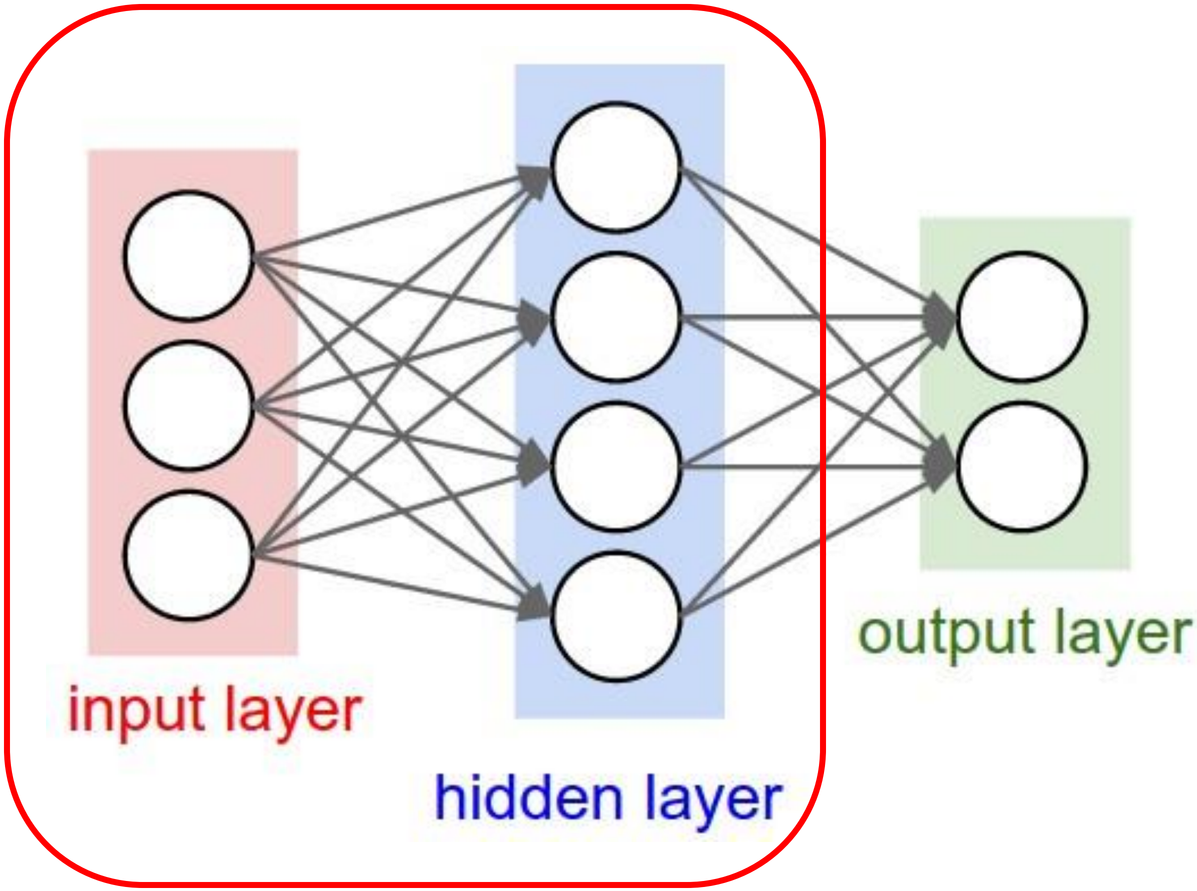
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d \left(y^{(d)} - \hat{y}^{(d)} \right)^2$$

Learning on Multi-Layer Perceptron



Difficult to calculate $\frac{\partial E}{\partial w_j}$

Learning on Multi-Layer Perceptron

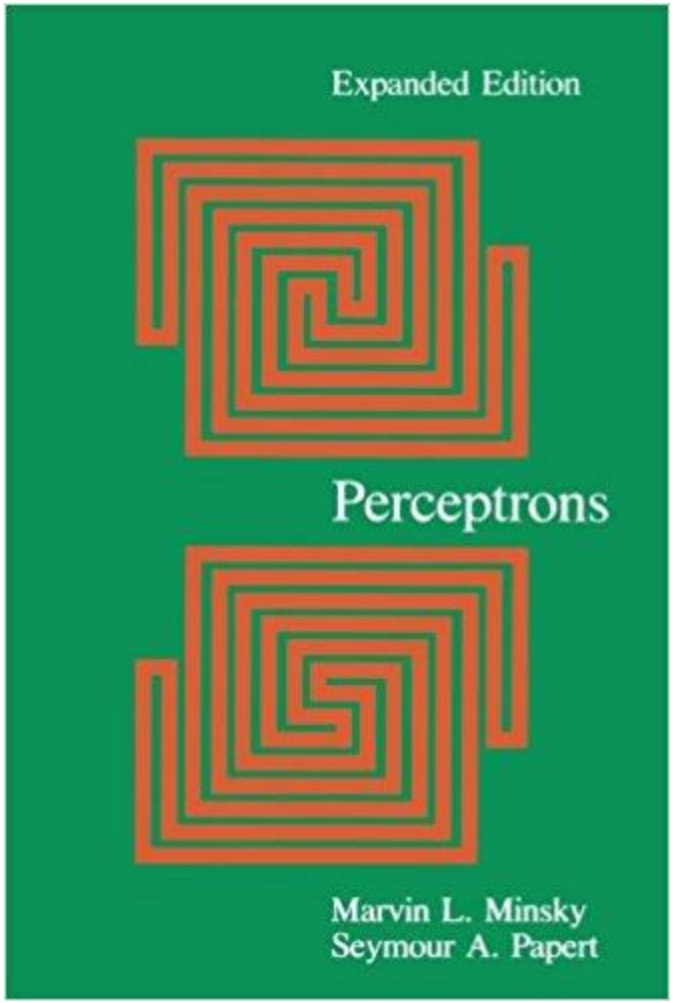


Unsolved problem
for 20 years

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d}\left(? - \hat{y}^{(d)}\right)^2$$

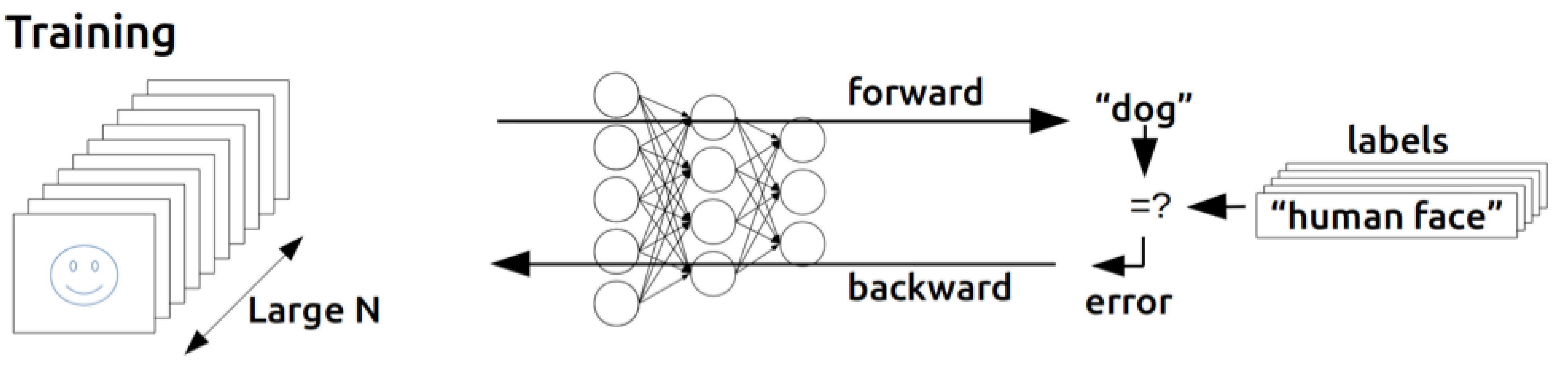
Limitation of Multi-layer Perceptron

By Marvin Minsky, founder of the MIT AI Lab.



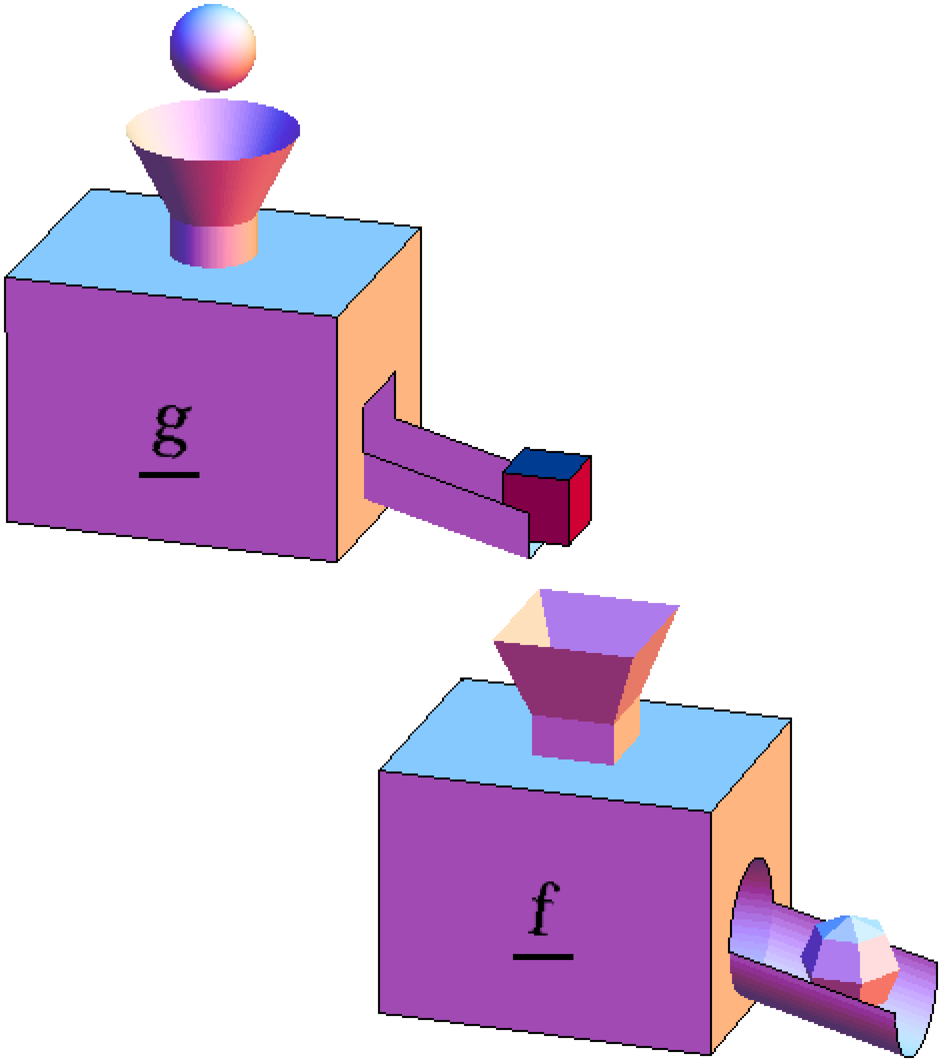
- We need to use MLP, multilayer perceptrons (multilayer neural nets)
- No one on earth had found a viable way to train MLPs good enough to learn such simple functions

Backpropagation
(1974, 1982 by Paul Werbos, 1986 by Hinton)



https://devblogs.nvidia.com/inference-next-step-gpu-accelerated-deep-learning/

Chain Rule



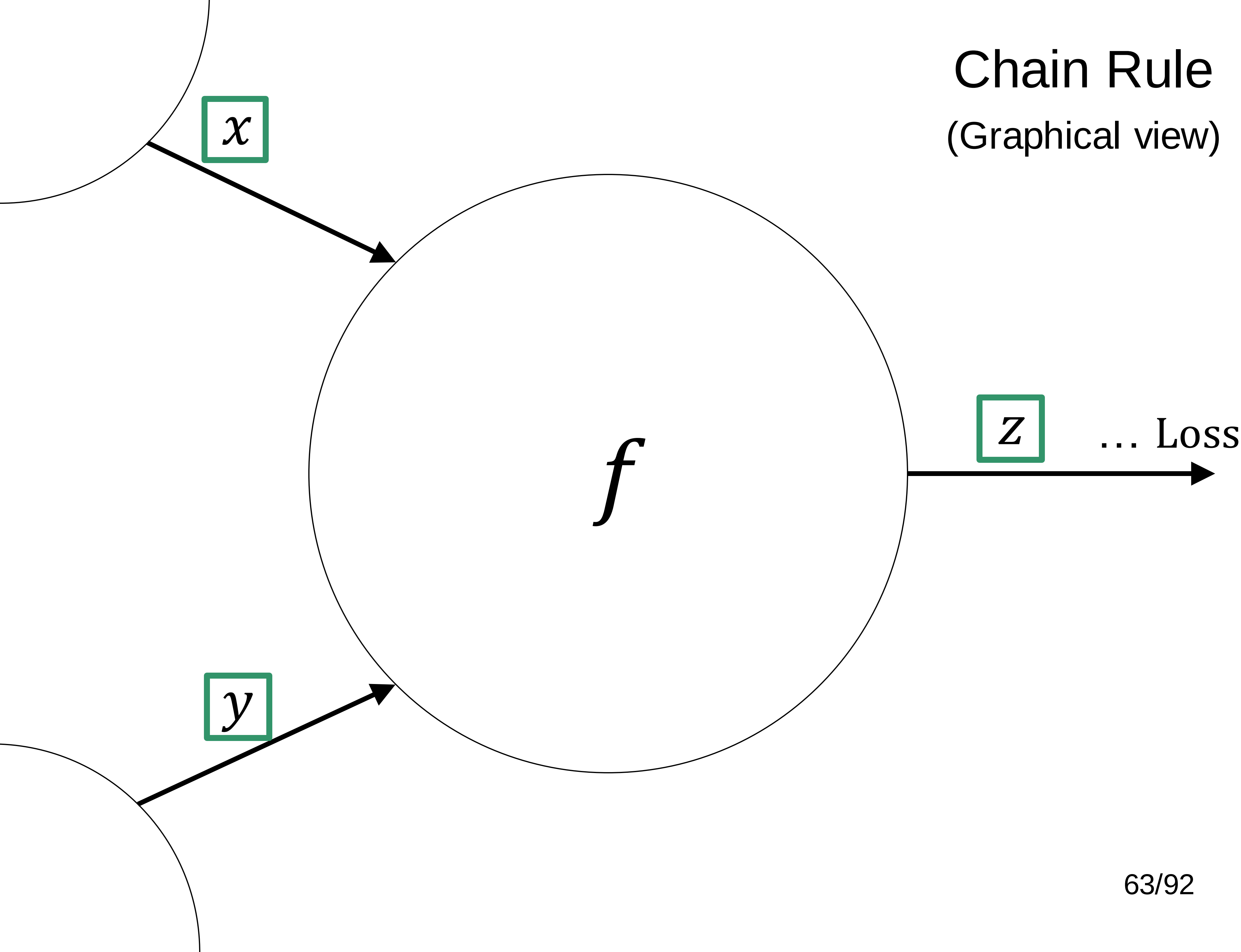
$$f = \underline{f}(g);\ g = \underline{g}(x)$$

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

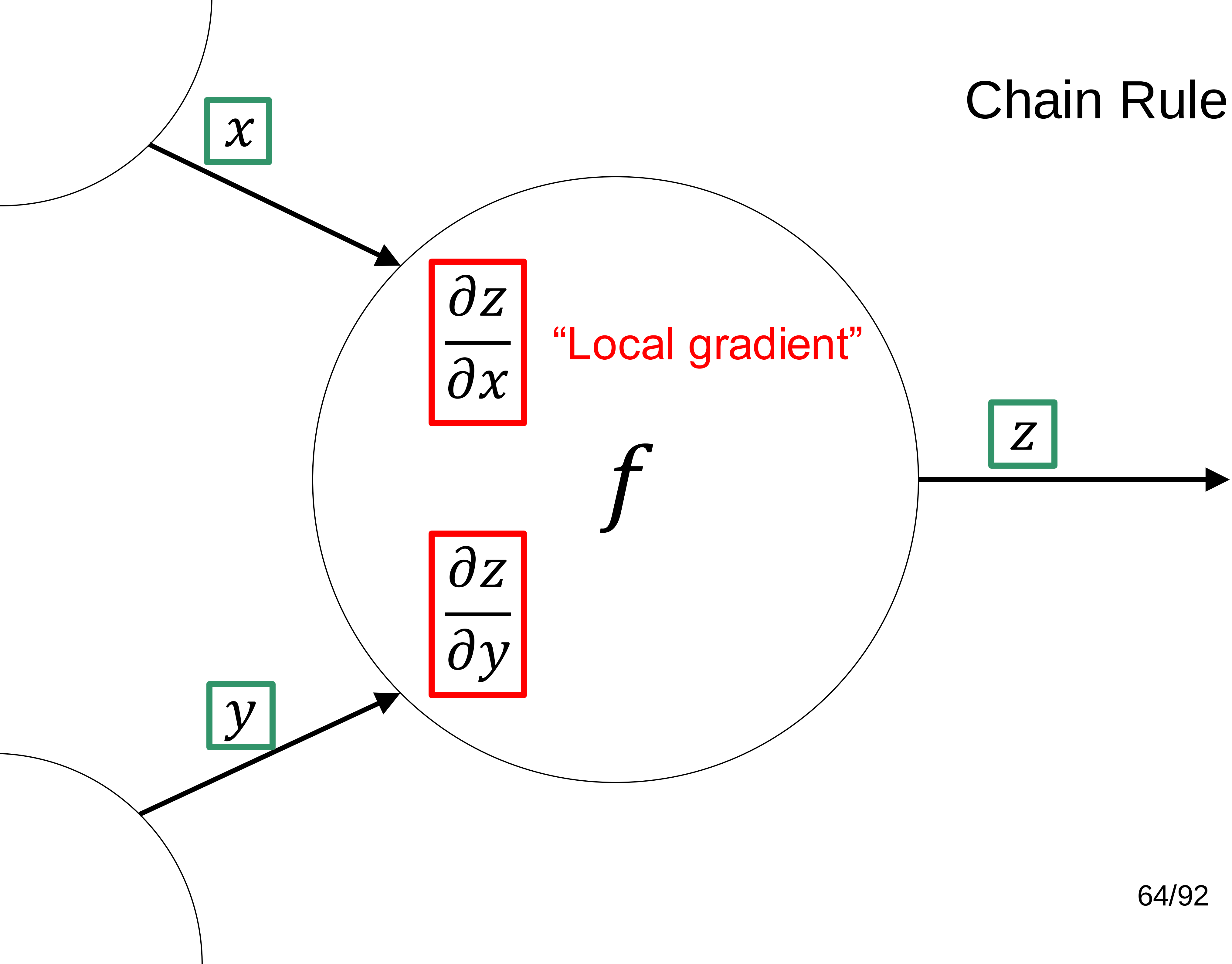
https://mathinsight.org/image/functino_machines_composed

Chain Rule

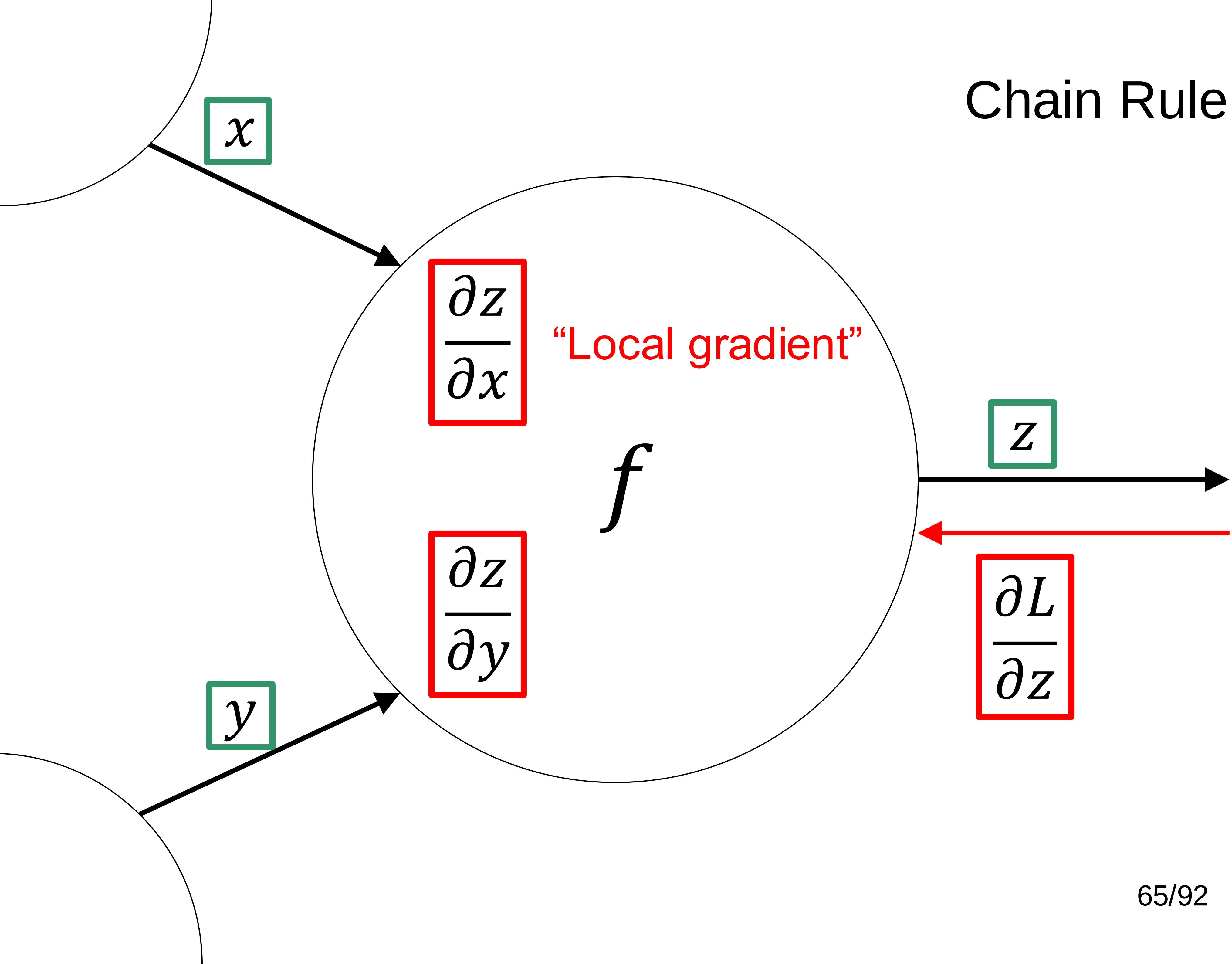
(Graphical view)



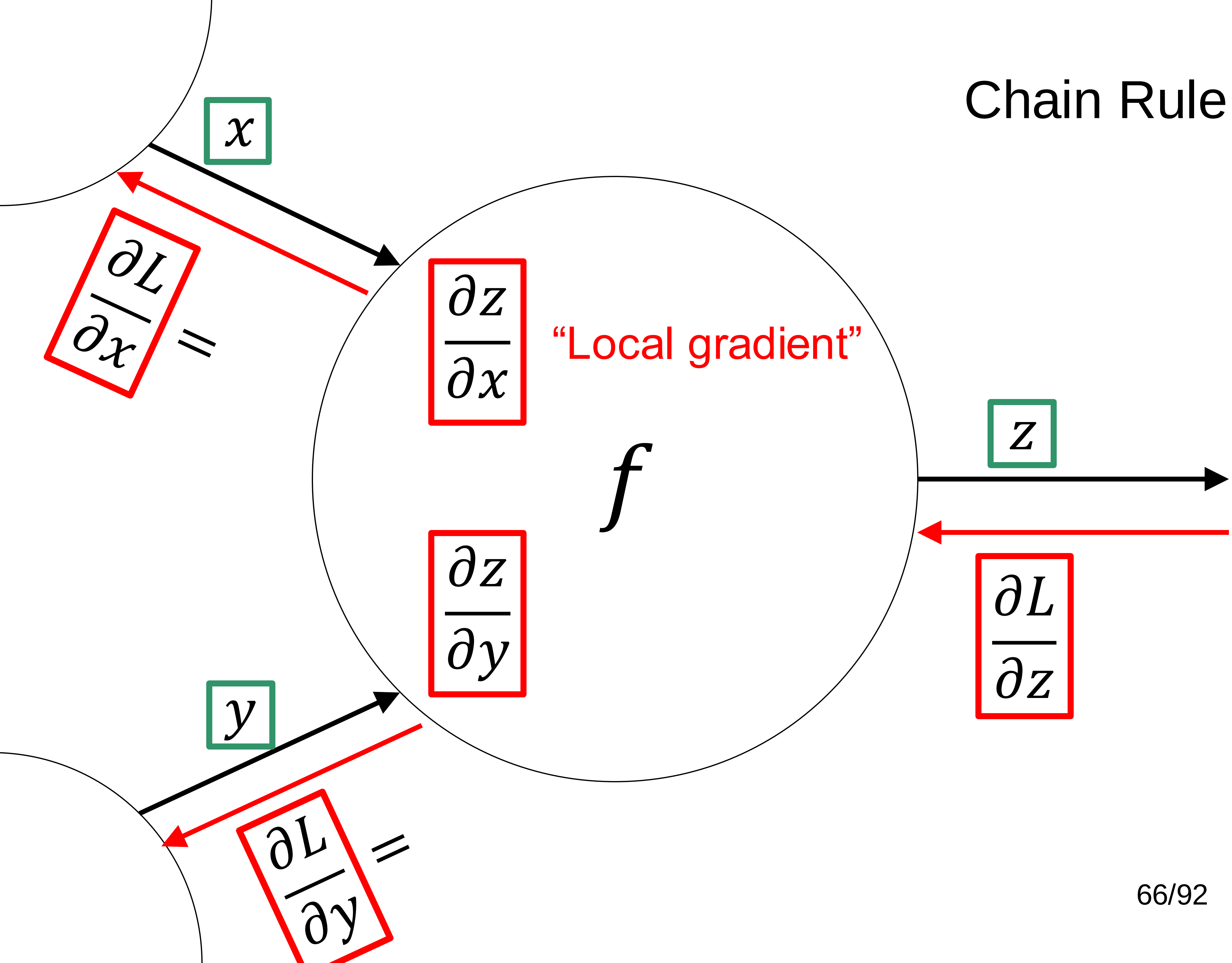
Chain Rule


x
y
z
f
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
"Local gradient"

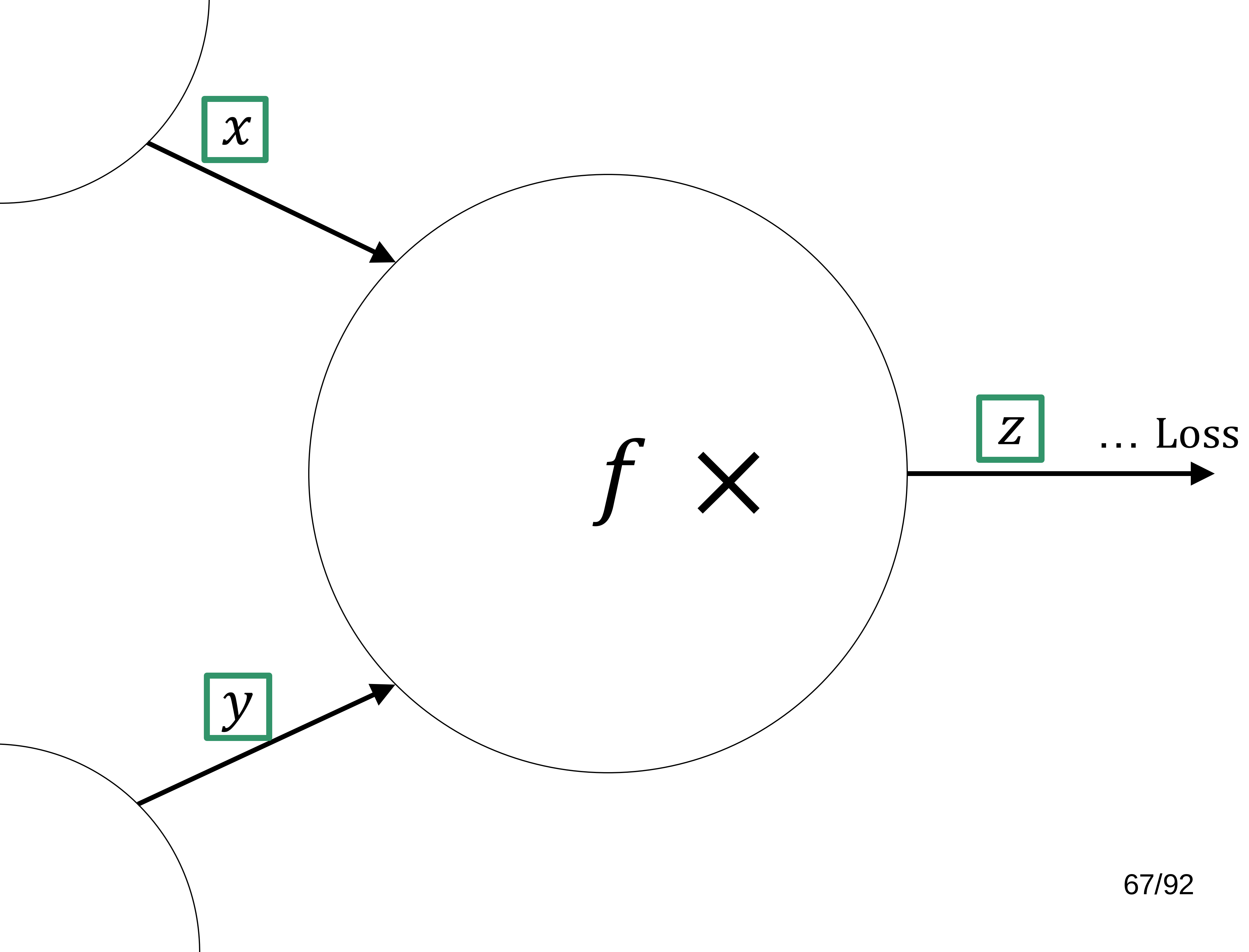
Chain Rule


x
$\frac{\partial z}{\partial x}$
“Local gradient”
f
z
$\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial y}$
y

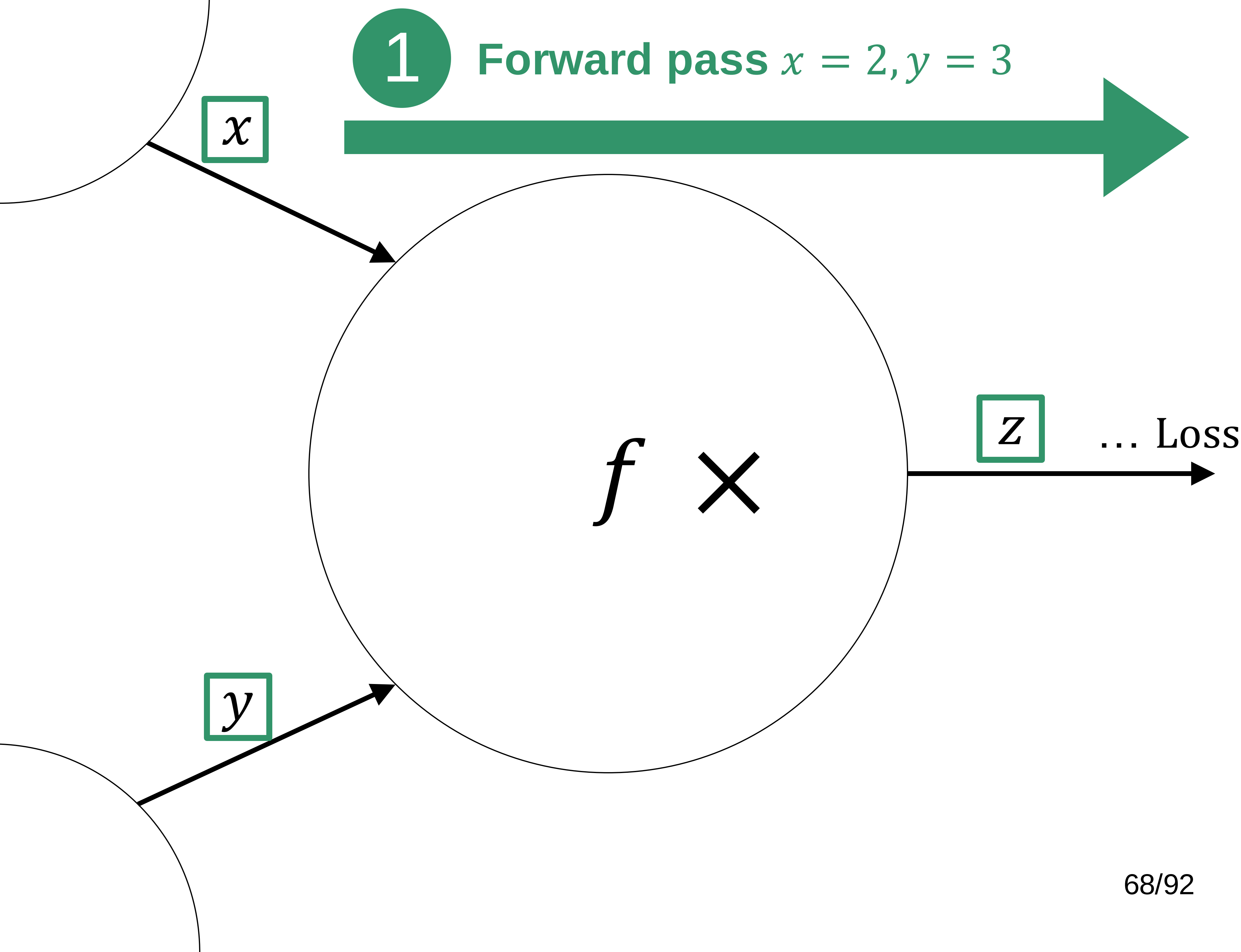
Chain Rule

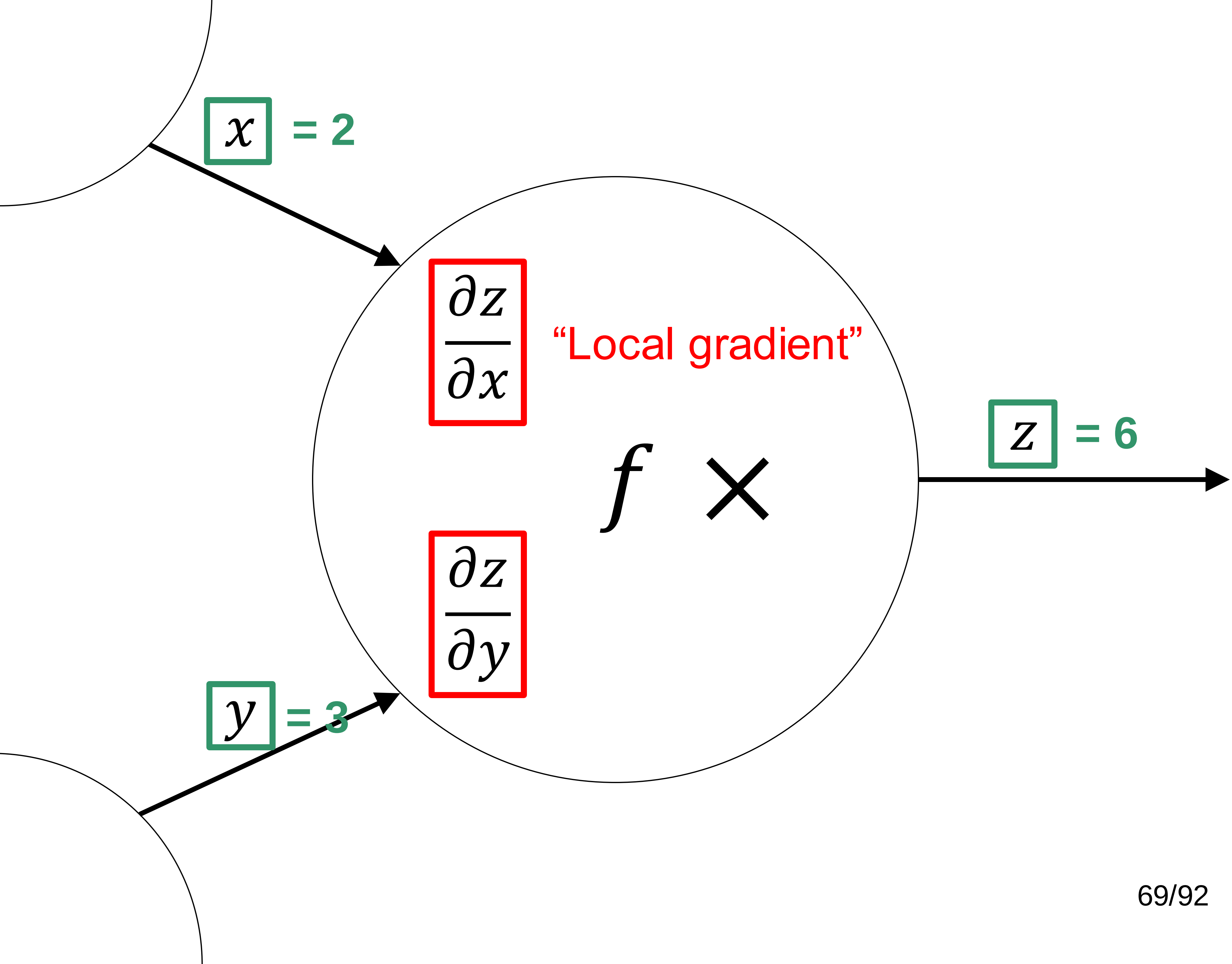


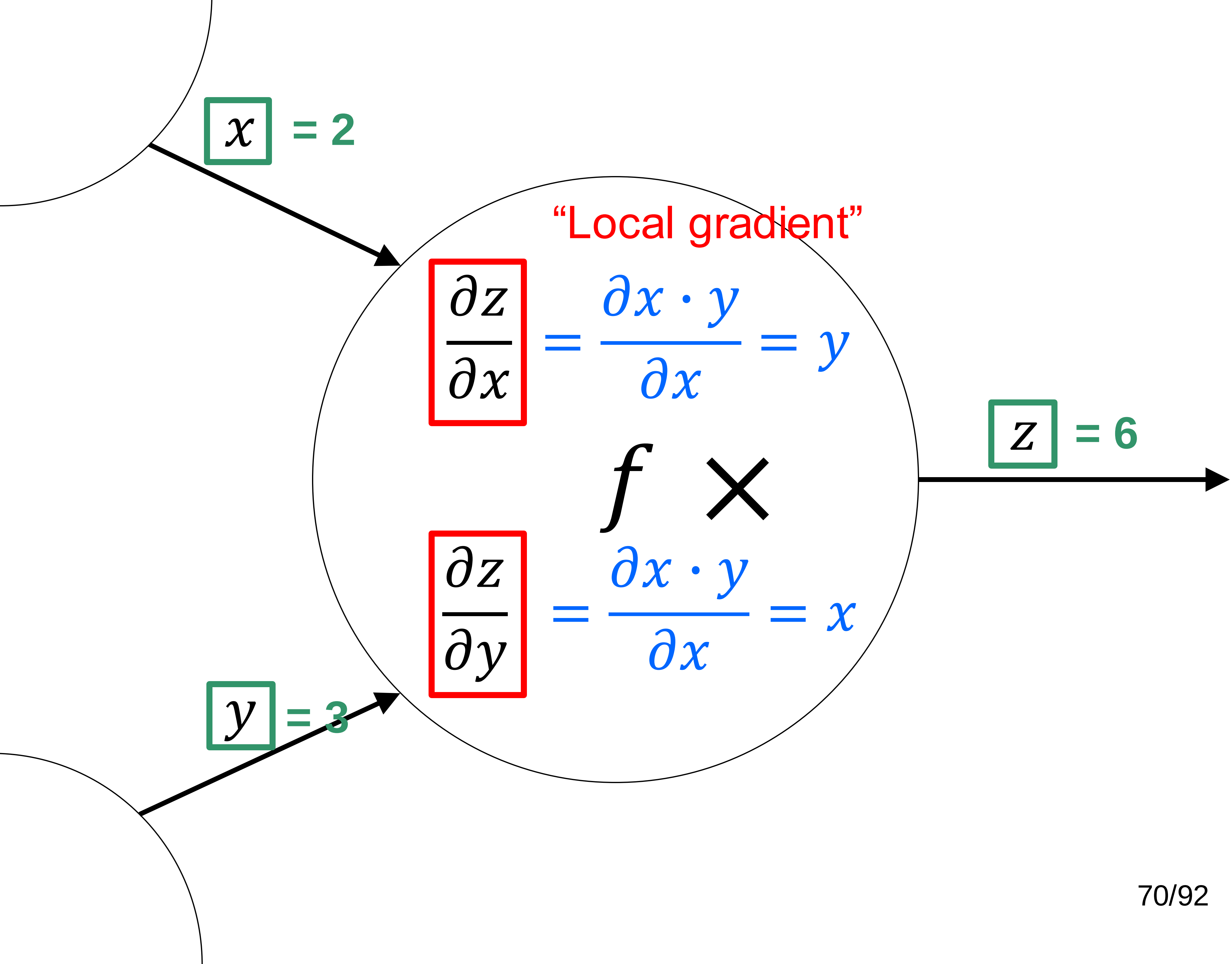
x
y
z
f
$\frac{\partial z}{\partial x}$
"Local gradient"
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial x} =$
$\frac{\partial L}{\partial y} =$

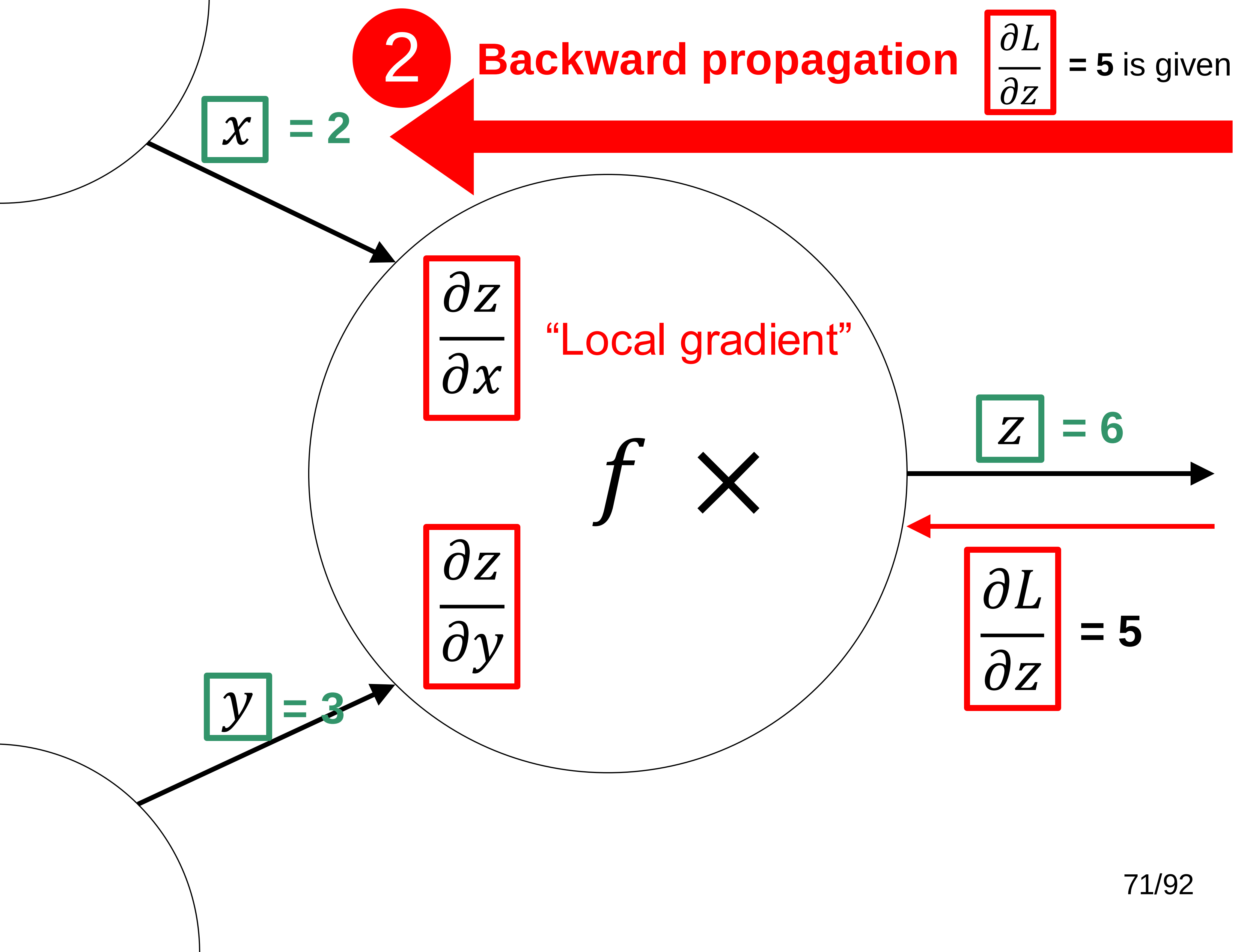

x
y
f ×
z
… Loss

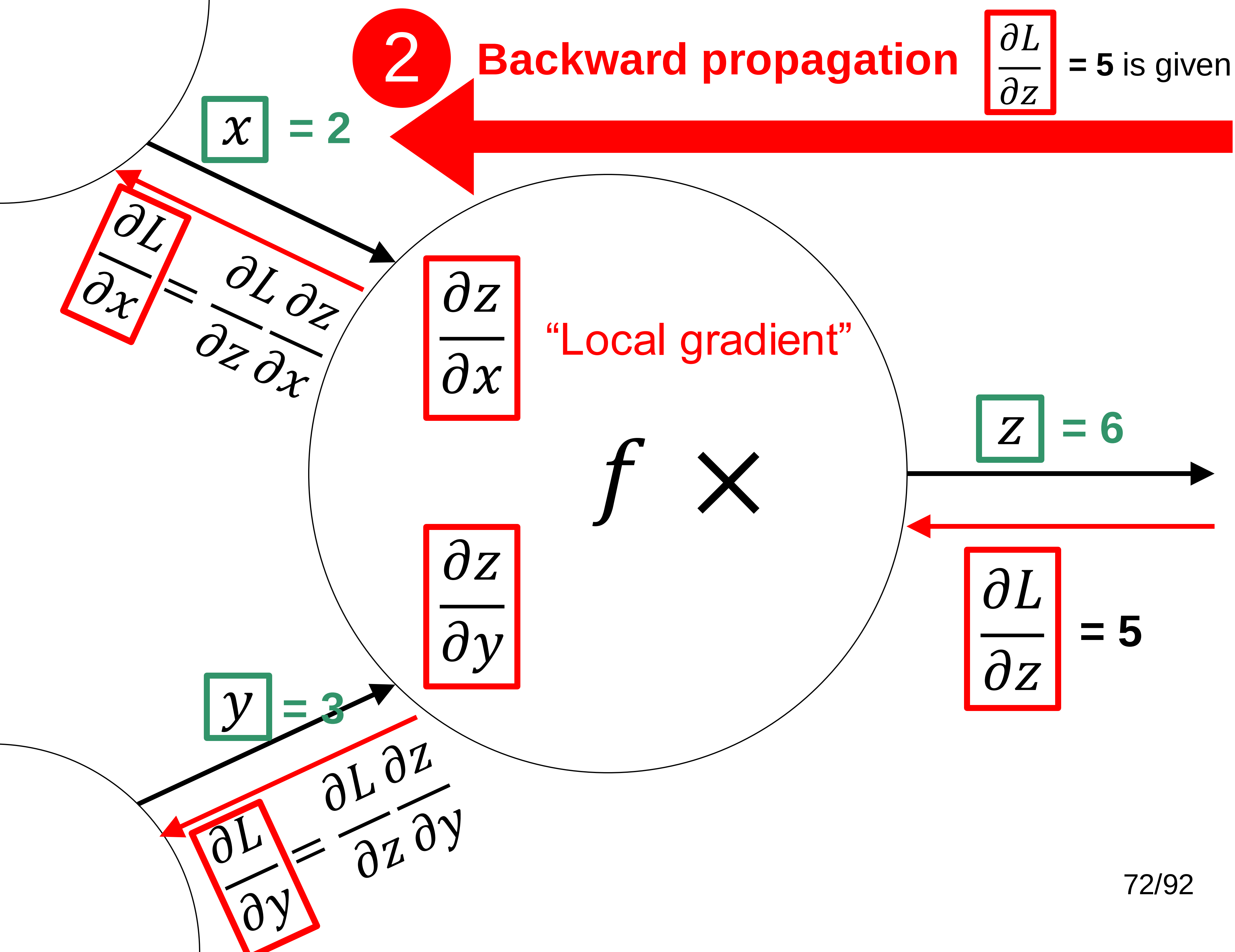
1 **Forward pass** $x = 2, y = 3$



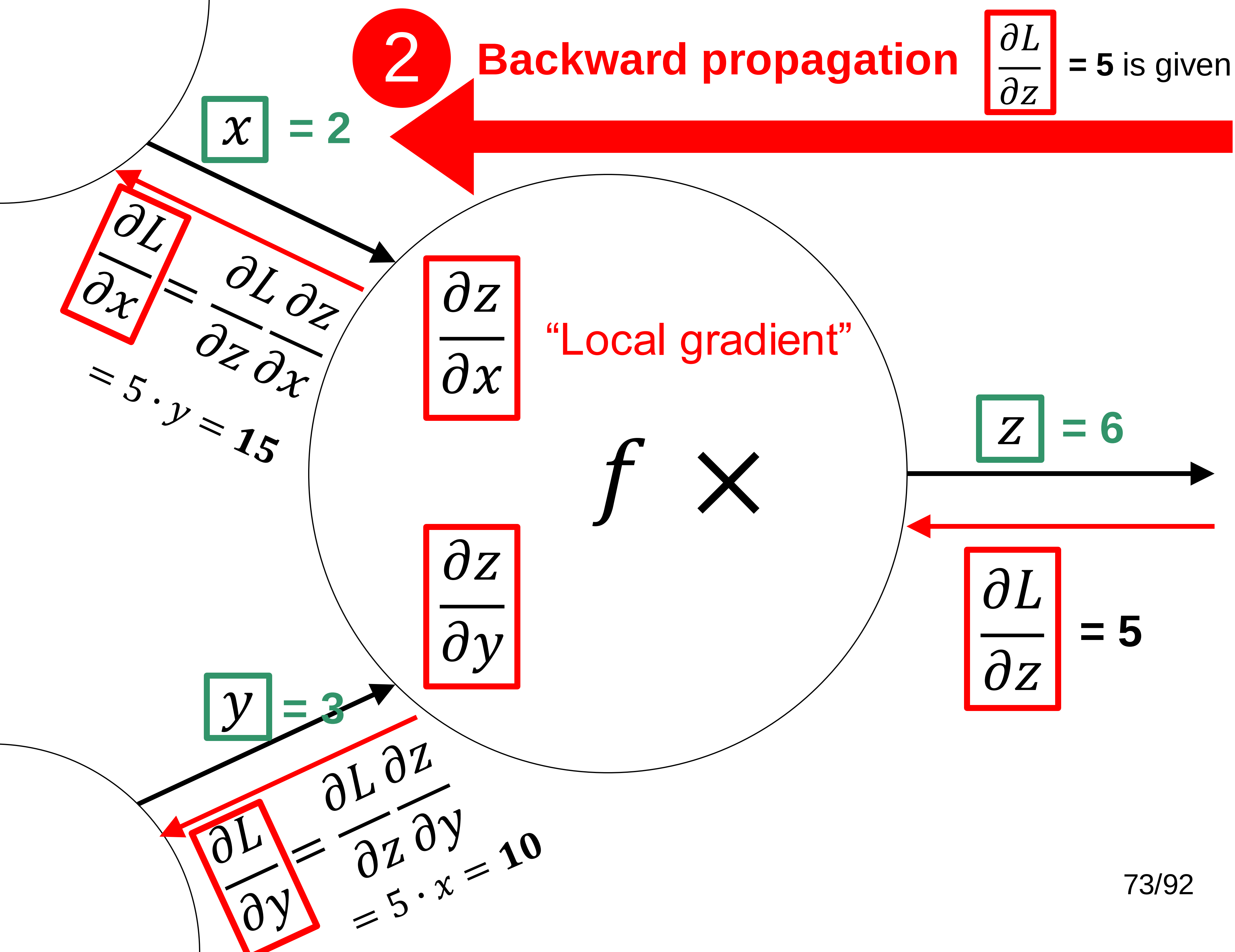

$\boxed{x}$ = 2
$\boxed{y}$ = 3
$\boxed{z}$ = 6
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
"Local gradient"
f ×


x = 2
"Local gradient"
$\frac{\partial z}{\partial x} = \frac{\partial x \cdot y}{\partial x} = y$
$f \times$
z = 6
$\frac{\partial z}{\partial y} = \frac{\partial x \cdot y}{\partial x} = x$
y = 3


2
Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given
x = 2
$\frac{\partial z}{\partial x}$
"Local gradient"
f ×
z = 6
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$ = 5
y = 3



2
Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given
x = 2
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
“Local gradient”
f ×
z = 6
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$ = 5
y = 3
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

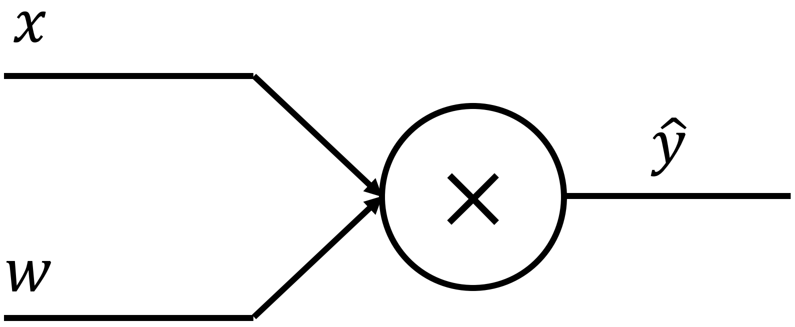

2
Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given
x = 2
y = 3
z = 6
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
"Local gradient"
f ×
$\frac{\partial L}{\partial z}$ = 5
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$= 5 \cdot y = \mathbf{15}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
$= 5 \cdot x = \mathbf{10}$

Computational graph

$$\hat{y} = x \times w$$

Computational graph

$$\hat{y} = x \times w$$

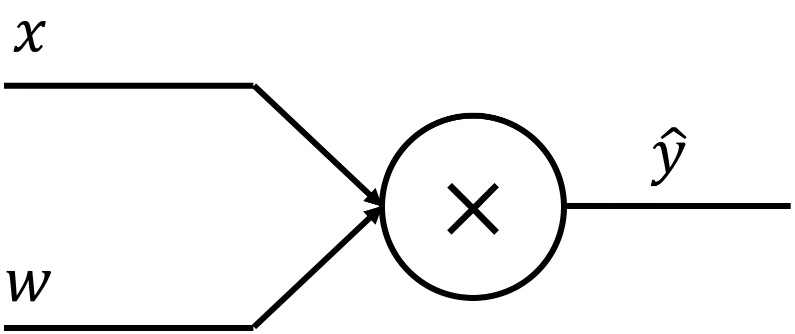


Computational graph

$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$

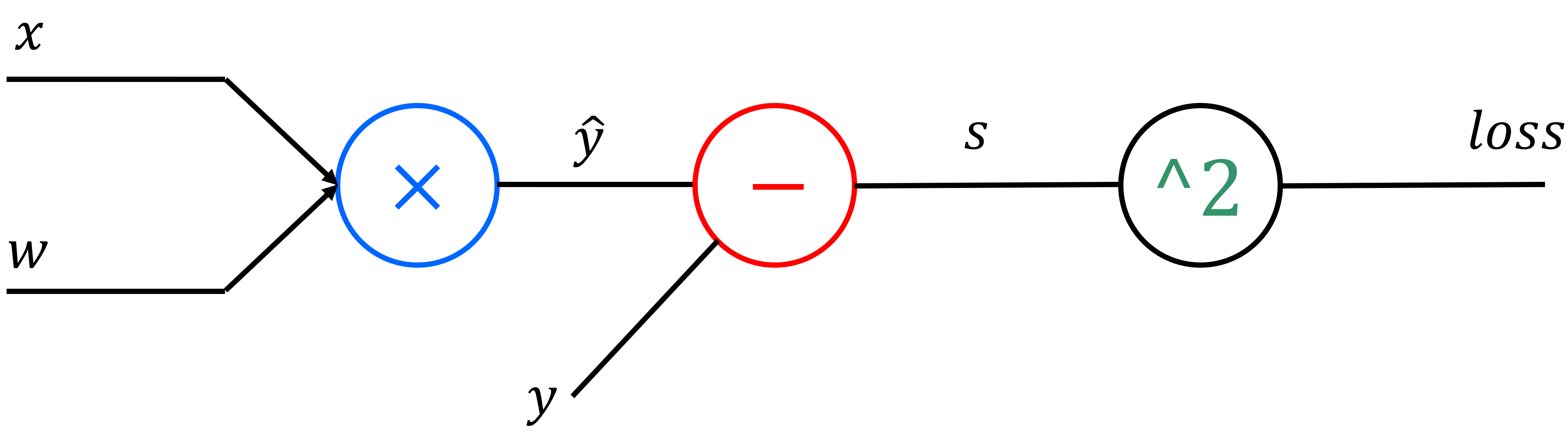
$$\frac{1}{2}\sum_{d \in D}\left(\hat{y}^{(d)} - y^{(d)}\right)^2$$

Our loss function



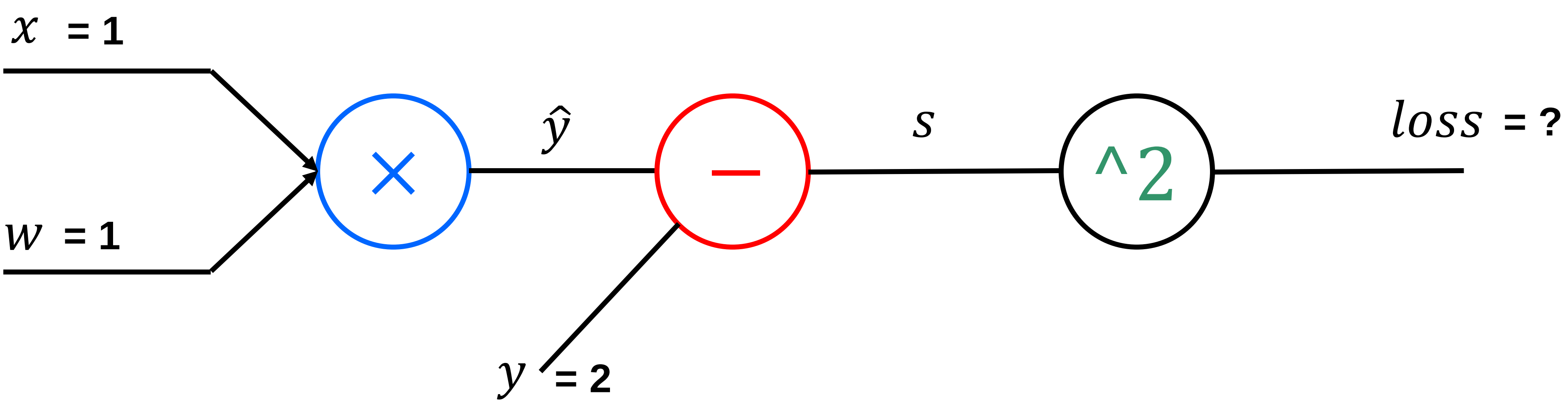
Computational graph

$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$



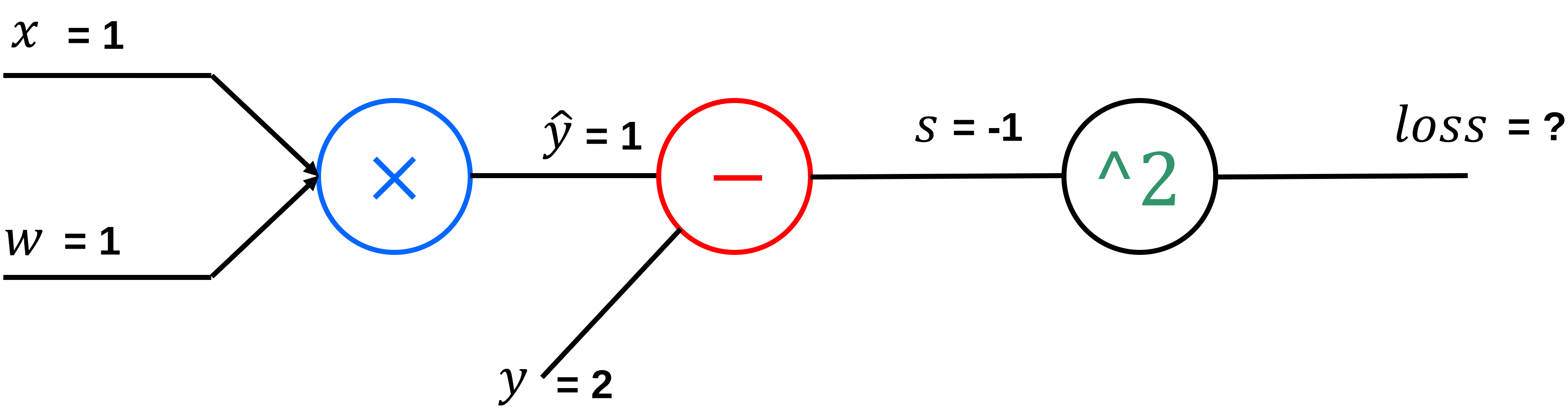
1 Forward pass $x = 1, y = 2$ where $w = 1$

$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$



1 Forward pass $x = 1, y = 2$ where $w = 1$

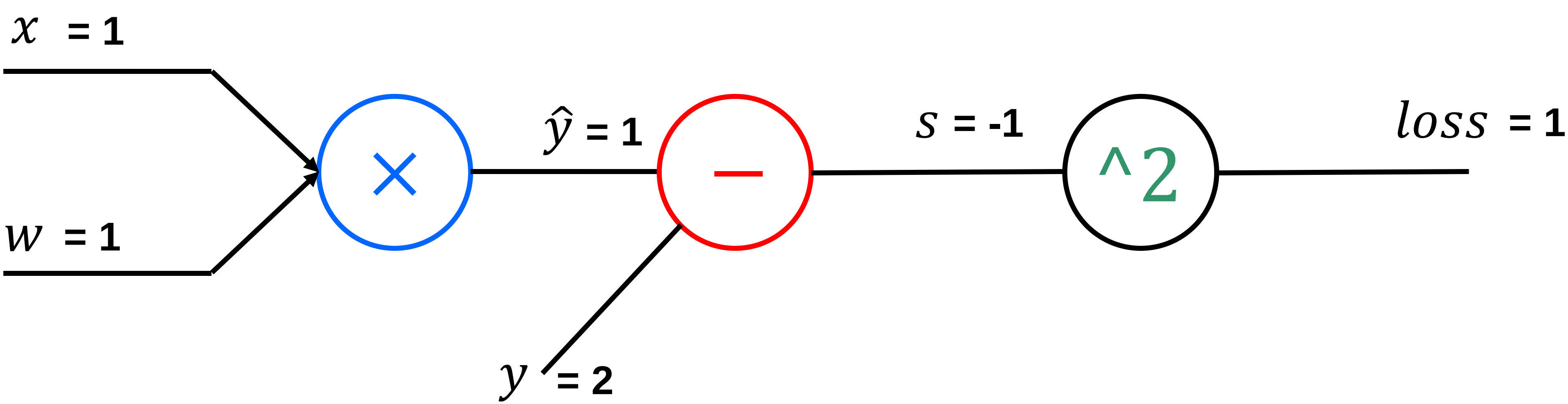
$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$



2

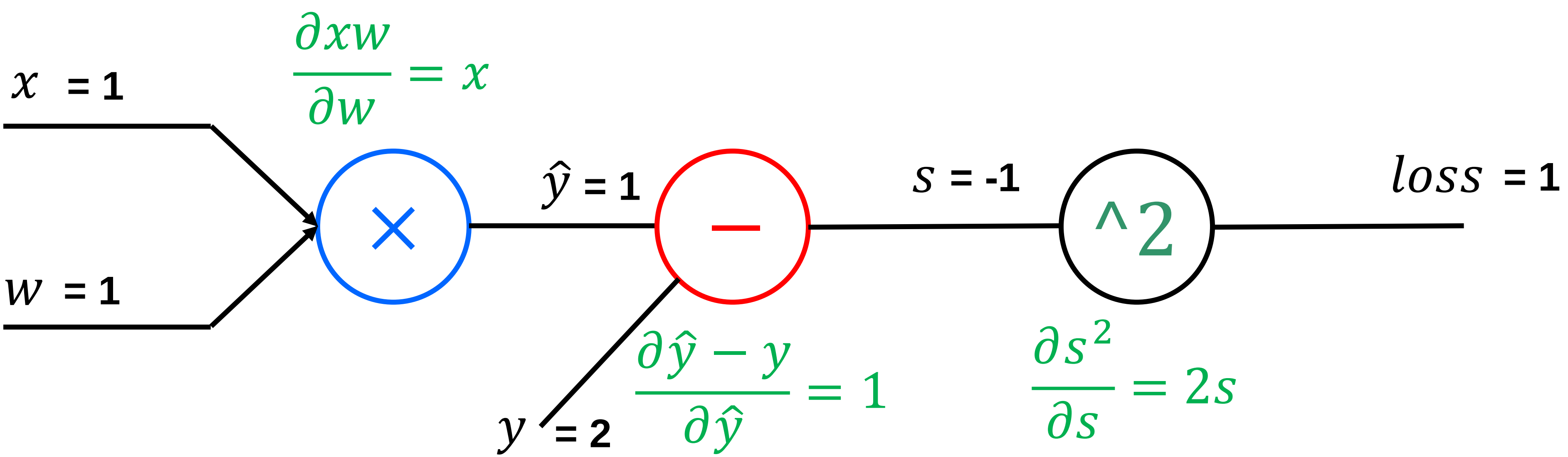
Backward propagation

$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$



Backward propagation

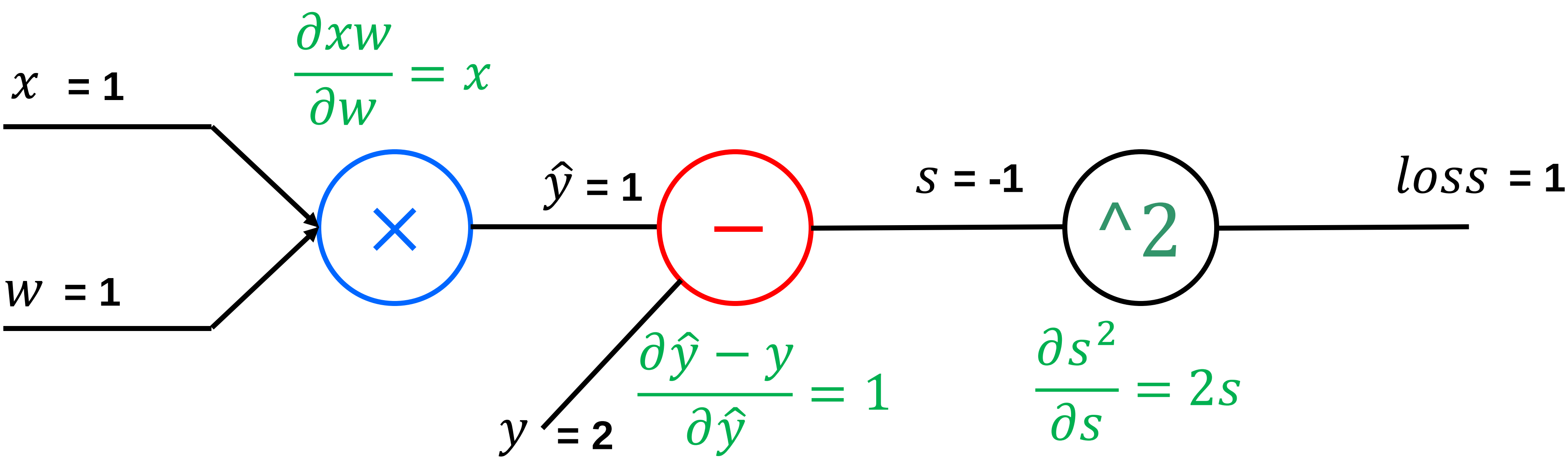
$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$



2 Backward propagation

$$\hat{y} = x \times w \qquad loss = (\hat{y} - y)^2 = (x \times w - y)^2$$

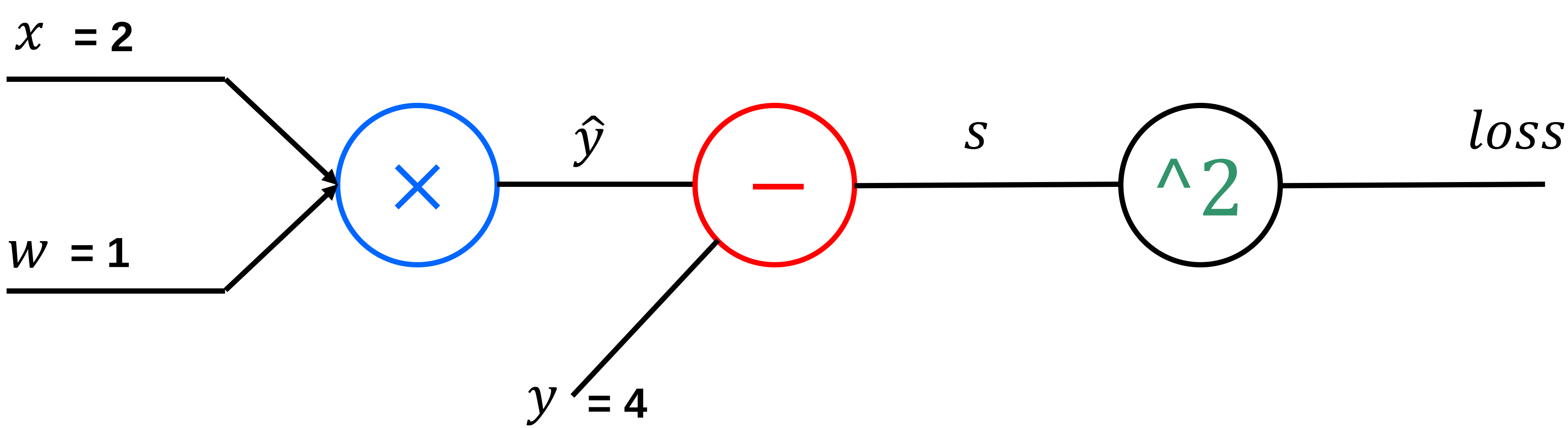
$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w} = -2 \cdot x = -2 \cdot 1 = -2$$



$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s} \frac{\partial s}{\partial \hat{y}} = -2 \cdot 1 = -2 \qquad \frac{\partial loss}{\partial s} = 2s = -2$$

$$\frac{\partial loss}{\partial w} = -2$$

Backpropagation: Exercise



$$\frac{\partial loss}{\partial w} = ?$$

Backpropagation: Summary

```
Initialize all weights to small random values.
repeat
    for each training example do
        Forward propagate the input features of the example
        to determine the MLP's outputs.
        Back propagate error to generate Δw_i for all weights Δw_i
        Update the weights using Δw_i
    end for
until stopping criteria reached.
```

Summary

- We learned what a perceptron and multilayer perceptron is
- We have some intuition about using gradient descent on an error function
- We know a learning delta rule for updating weights in order to minimize the error: $\Delta w_i = -\eta \times \frac{\partial E}{\partial w_i}$
- We know activation function for non-linearity
- We can use this rule to learn an MLP using the backpropagation algorithm

Optimizer: How to Optimize?

Gradient Descent (GD) vs. Stochastic GD (SGD)

Objective to Minimize:

$$E(\mathbf{w}) \equiv \frac{1}{D}\sum_{d \in D} E^{(d)}(\mathbf{w})$$

GD:

while True:

$$\underbrace{\mathbf{w}^{new} \leftarrow \mathbf{w}^{old} - \eta \frac{\partial}{\partial \mathbf{w}} \frac{1}{D} \sum_{d \in D} E^{(d)}(\mathbf{w})}_{O(N)}$$

SGD:

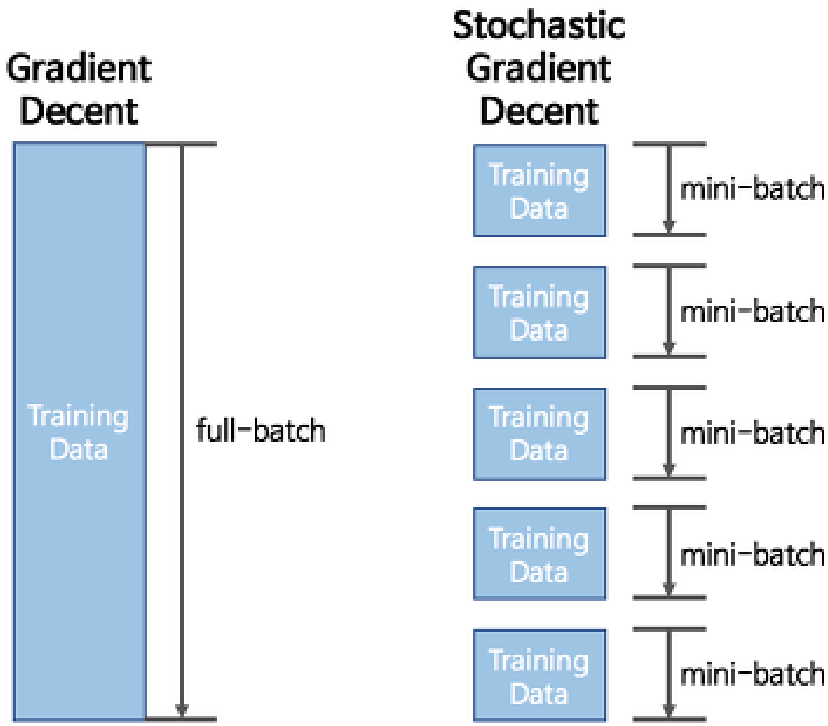
while True:

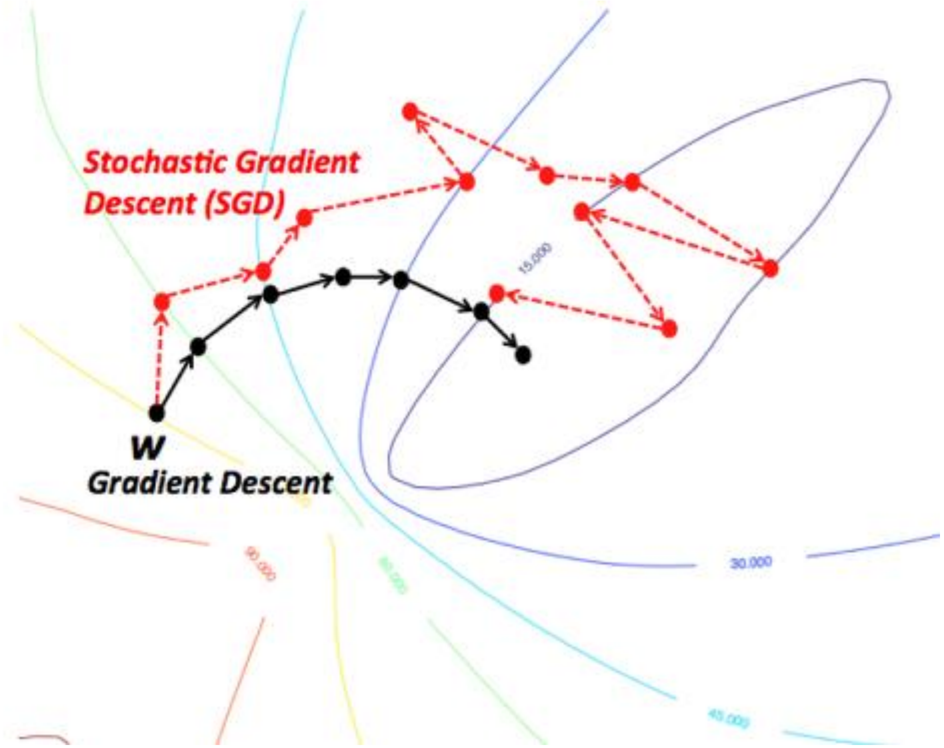
$$d \leftarrow \mathit{mini\ batch(1,N)}$$

$$\underbrace{\mathbf{w}^{new} \leftarrow \mathbf{w}^{old} - \eta \frac{\partial E^{(d)}(\mathbf{w})}{\partial \mathbf{w}}}_{O(d)}$$

Optimizer: How to Optimize?

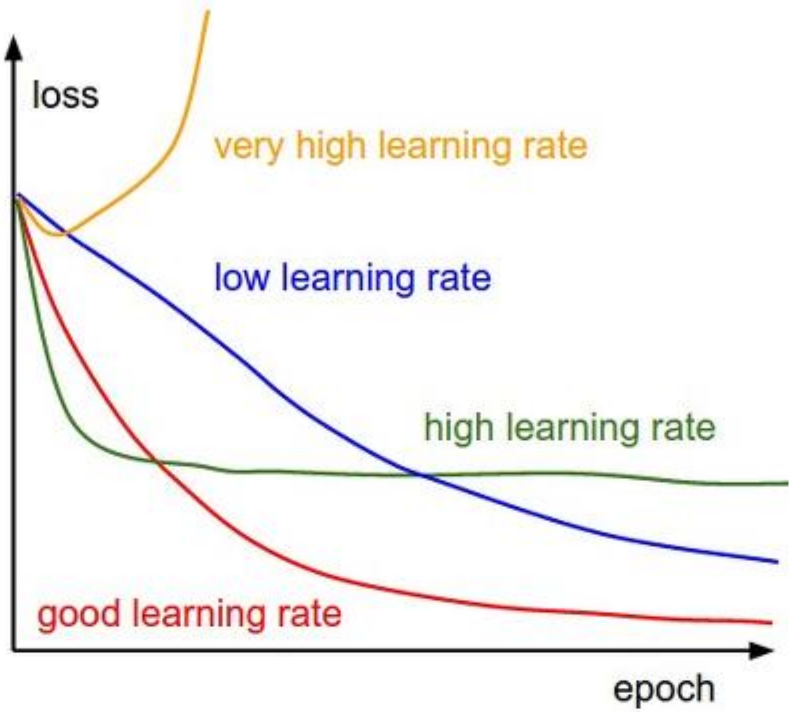
Gradient Descent vs. SGD

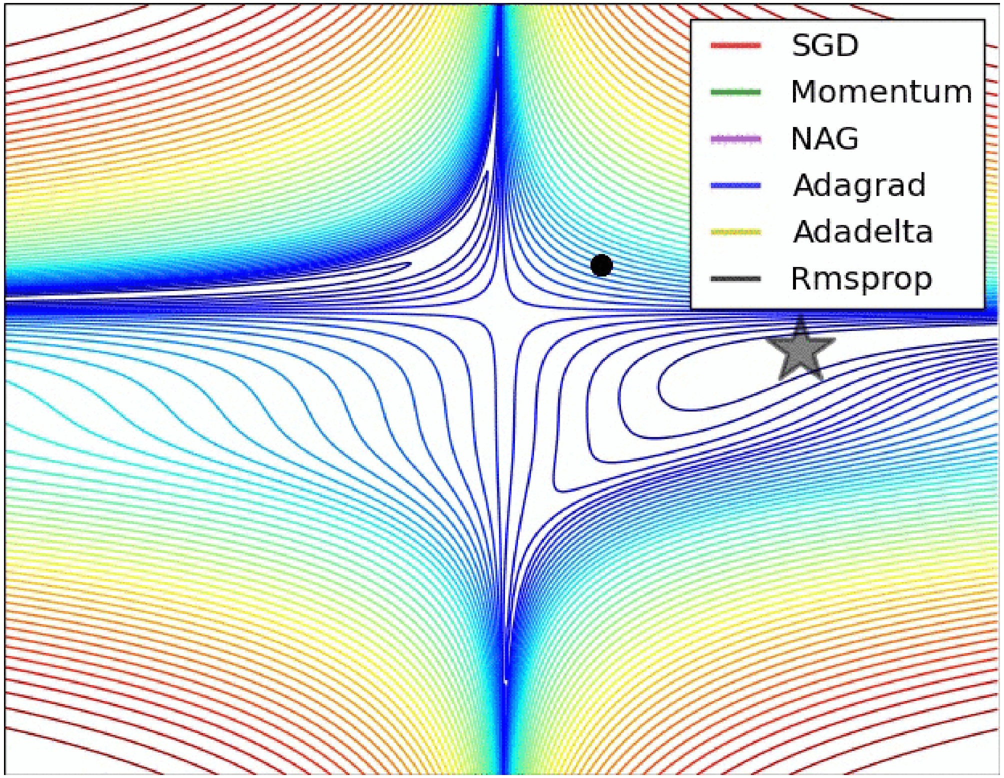




GD vs. SGD

Limitations of SGD and its Solutions





http://ruder.io/optimizing-gradient-descent/

Tutorial on Neural Network: Zero to All !

- Perceptron from scratch
  - Based and basic algebra
- Perceptron based on pytorch
- Multi-layer perceptron

Step	Method	Data	Model	Forward	Loss	Backward	Update
1	Perceptron	x	x	x	x	x	x
2		x	x	x	x	o	x
3		x	o	o	o	o	o
4	MLP	o	o	o	o	o	o

Deep Neural Network Toolbox

Artificial Intelligence and Machine Learning SkillsFuture Courses and Training

- Why PYTORCH?
  - More pythonic
    - Flexible, intuitive and cleaner code, and easy to debug
- More neural networkic
  - Write code as the network works and forward/backward

Multi-Layer Perceptron: Programming

1. Training data

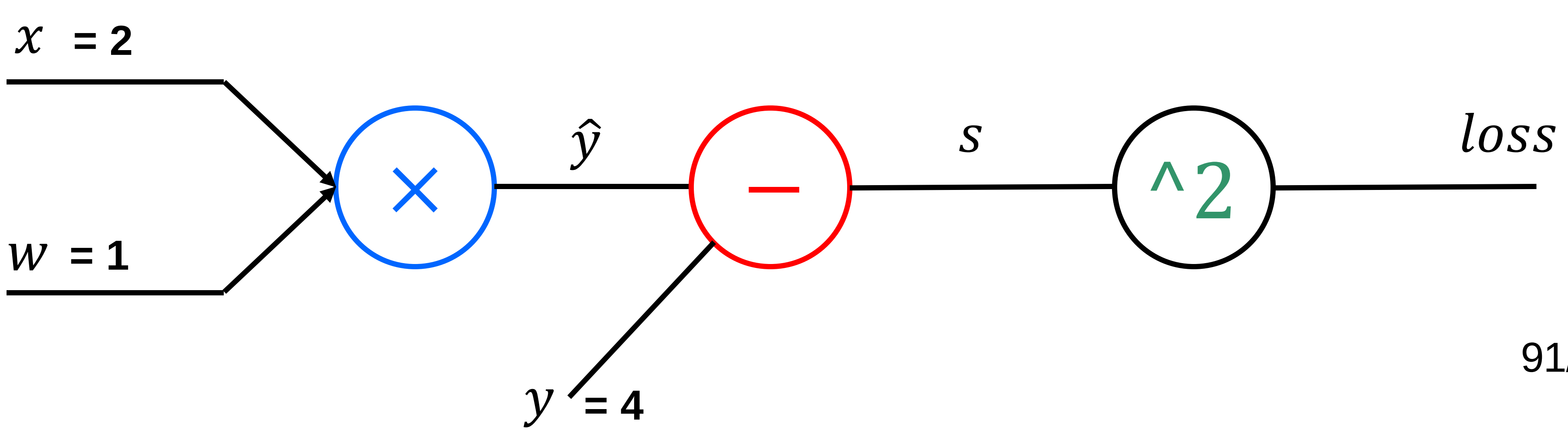
x	y
1	2
2	4
3	6

2. Model (forward pass)

$$\hat{y} = x \times w$$

3. Loss function

$$loss = (\hat{y} - y)^2 = (x \times w - y)^2$$



Multi-Layer Perceptron: Programming

1. Training data

x	*y*
1	2
2	4
3	6

2. Model (forward pass)

$$\hat{y} = x \times w$$

3. Loss function

$$loss = (\hat{y} - y)^2 = (x \times w - y)^2$$

Learning Process (Very important !!!!)

Forward → Loss → Backward → Update