DEEPHEALTH

EDDL

Deep Learning with EDDL

Winter School 24/01/2022

Contents

Monday 24 and Tuesday 25
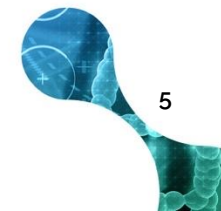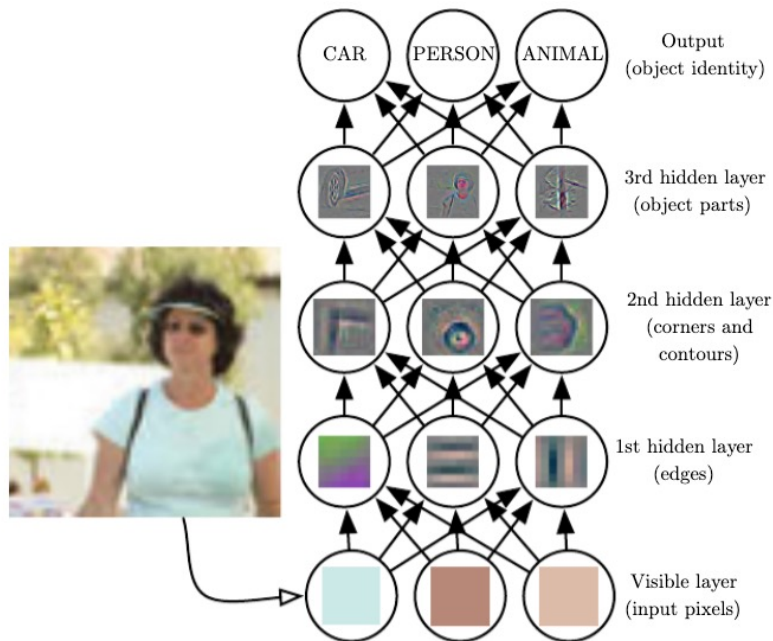
What is Deep Learning?
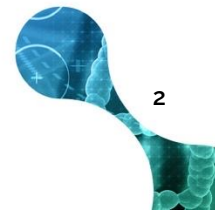
What is Deep Learning?

- Machine learning models based on neural networks:
  - Non-linear models
  - Learning by iteratively showing pairs (input,target) examples
  - Topology defined from know-how, hand-crafted
  - Learn the weights (parameters) by **gradient descent** of a particular **loss function**, iterative procedure
- Deep means that we stack lot of layers (20, 50, 100, 1000)
- Historically neural nets failed to learn with more than (let's say) 10 layers
- Why we need to go deep ? ⟶ Representation Learning
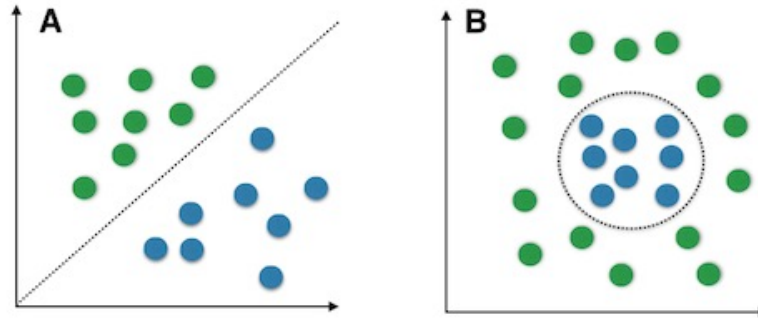
What is Deep Learning?

- Going deep

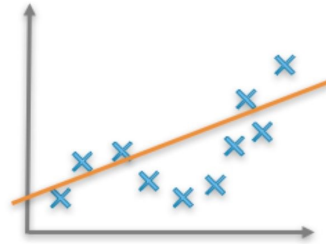# What is Deep Learning?

- Non-linear models

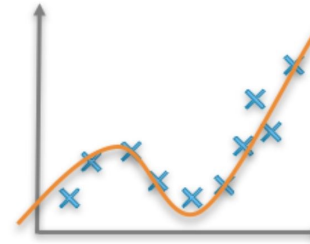Linear vs. nonlinear problems

What is Deep Learning?

- Non-linear models

Linear function   Non-linear function

Best fit linear and non-linear models

What is Deep Learning?

- Learning by iteratively showing pairs (input,target) examples
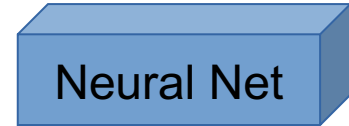
Training set

Input   Target

"Cat"

Neural Net

What is Deep Learning?

- Learning by iteratively showing pairs (input,target) examples

Training set

Input   Target

"Dog"   Neural Net
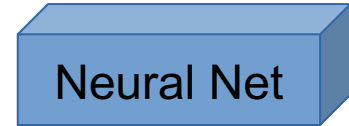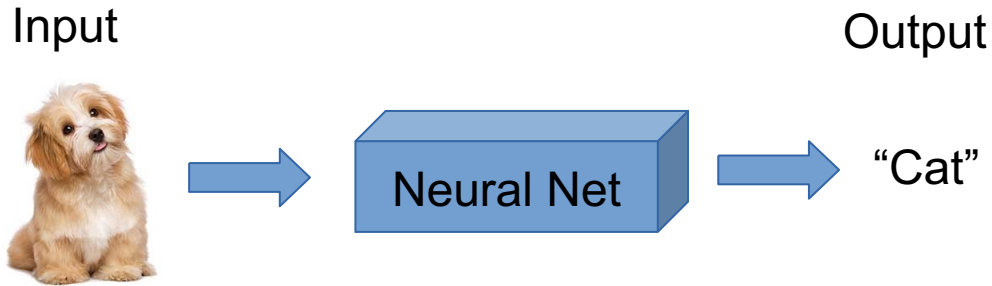
What is Deep Learning?

- Learning by iteratively showing pairs (input,target) examples

Input

Output

Neural Net

"Cat"

What is Deep Learning?

- Learning by iteratively showing pairs (input,target) examples

Input   Output   Target

Neural Net → "Cat" ⟷ "Dog"

Loss

What is Deep Learning?

- Learn the weights (parameters) by **gradient descent** of a particular **loss function**, iterative procedure

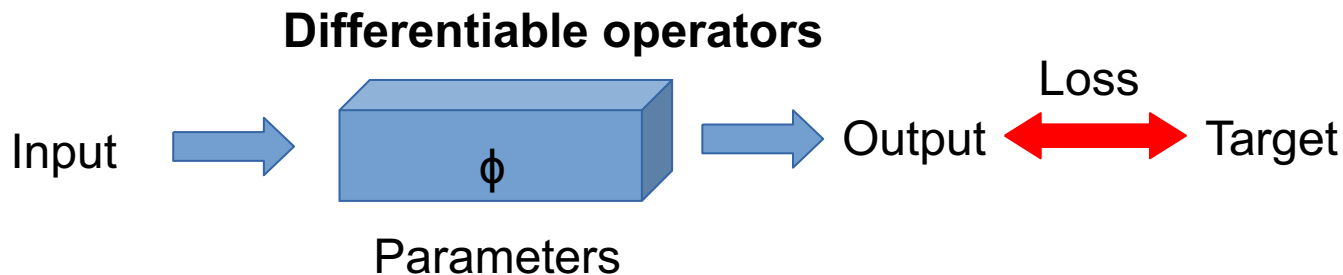Input $\rightarrow$ ϕ $\rightarrow$ Output $\longleftrightarrow$ Target

Loss

Parameters

What is Deep Learning?

- Learn the weights (parameters) by **gradient descent** of a particular **loss function**, iterative procedure

Differentiable operators

Input → ϕ → Output ←→ Target

Loss

Parameters

What is Deep Learning?

- Learn the weights (parameters) by **gradient descent** of a particular **loss function**, iterative procedure

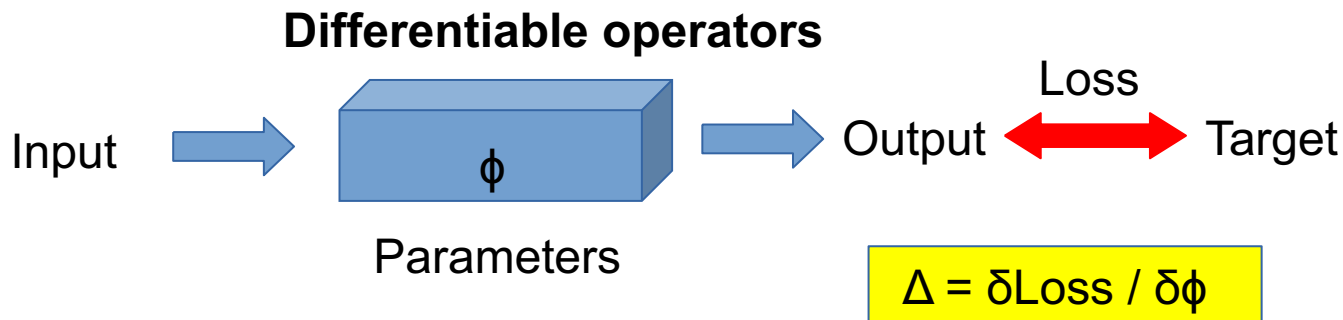**Differentiable operators**

Input → ϕ → Output ↔ Target
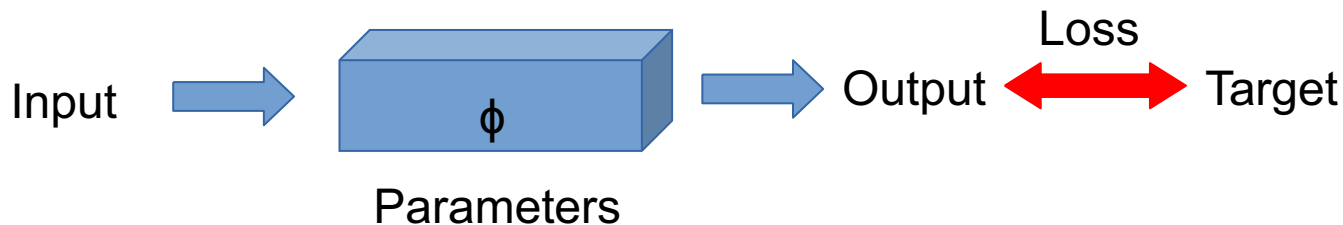
Loss

Parameters

$$\Delta = \delta Loss / \delta \phi$$

What is Deep Learning?

- Learn the weights (parameters) by **gradient descent** of a particular **loss function**, iterative procedure

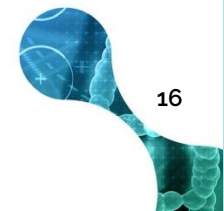$$\Delta = \delta Loss / \delta\phi$$

$$\phi = \phi - \mu\Delta$$

- μ is what we call learning rate

What is Deep Learning?

- Topology defined from know-how

  - Raw Data : Dense Layers
  - Temporal: Recurrent Layers
  - Images: Covolutional Layers
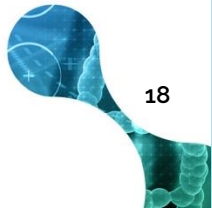
What is Deep Learning?

- Topology defined from know-how

    - Raw Data : Dense Layers
    - Temporal: Recurrent Layers
    - Images: Covolutional Layers
        - Classification: resnet, densenet,
        - Segmentation: U-net
        - Detection: FasterRCNN, Yolo, SSD
        - Pixel annotation: MaskRCNN
        - Image Generation: DCGAN, CylceGan
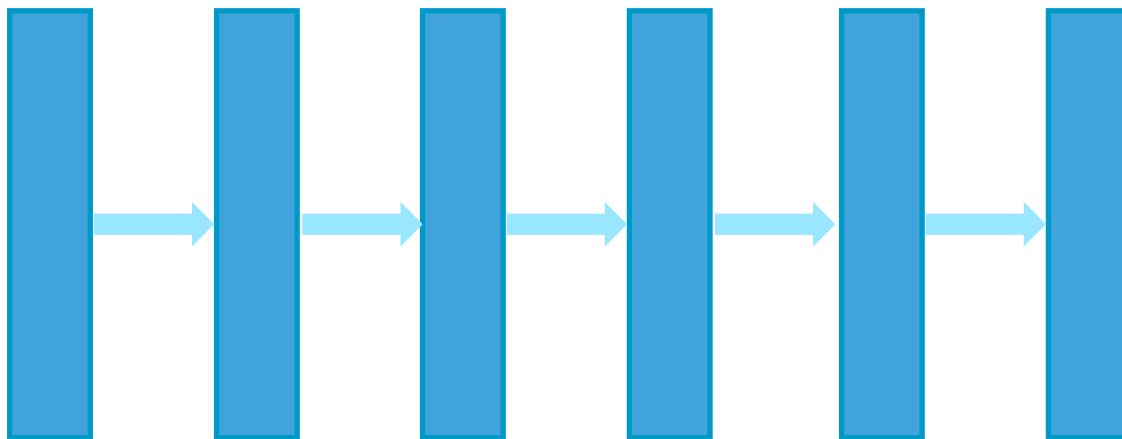   - Fine grain classification: Bilinear CNN
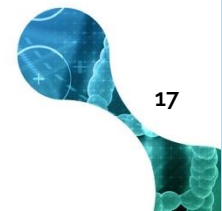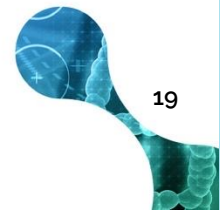
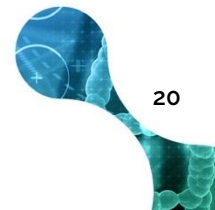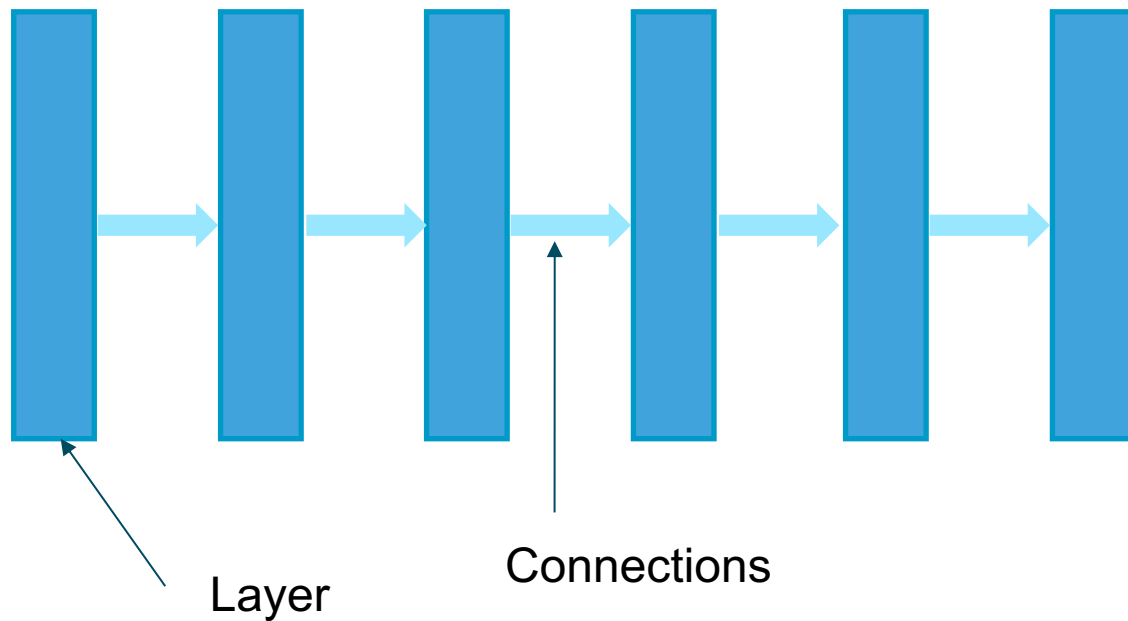# What is a Neural Network?

What is a Neural Network?

A connectionist model
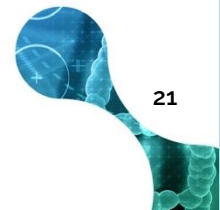
What is a Neural Network?

Layer

Connections

What is a Neural Network?
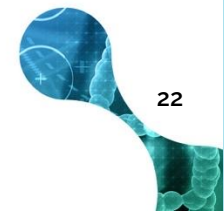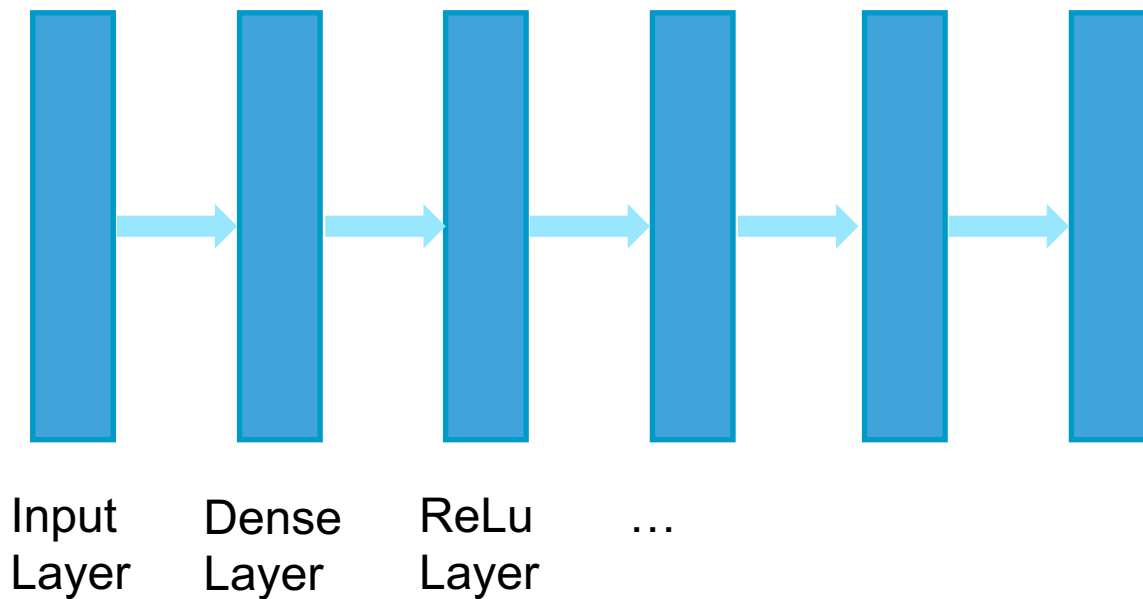
Layer: Operation over tensors (1D, 2D, 3D …)

Connection: the output tensor of a layer (result) is used as input tensor of the following layer

We would refer "x" as input and "y" as output

What is a Neural Network?

Input
Layer

Dense
Layer

ReLu
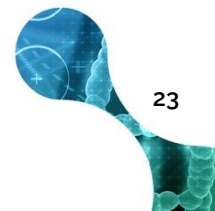Layer

…

What is a Neural Network?

Input Layer: Receive the input examples (images of cats)

Dense Layer: Is a **parametric** layer with the following operation:

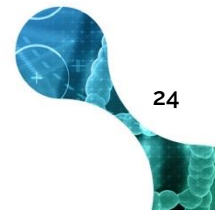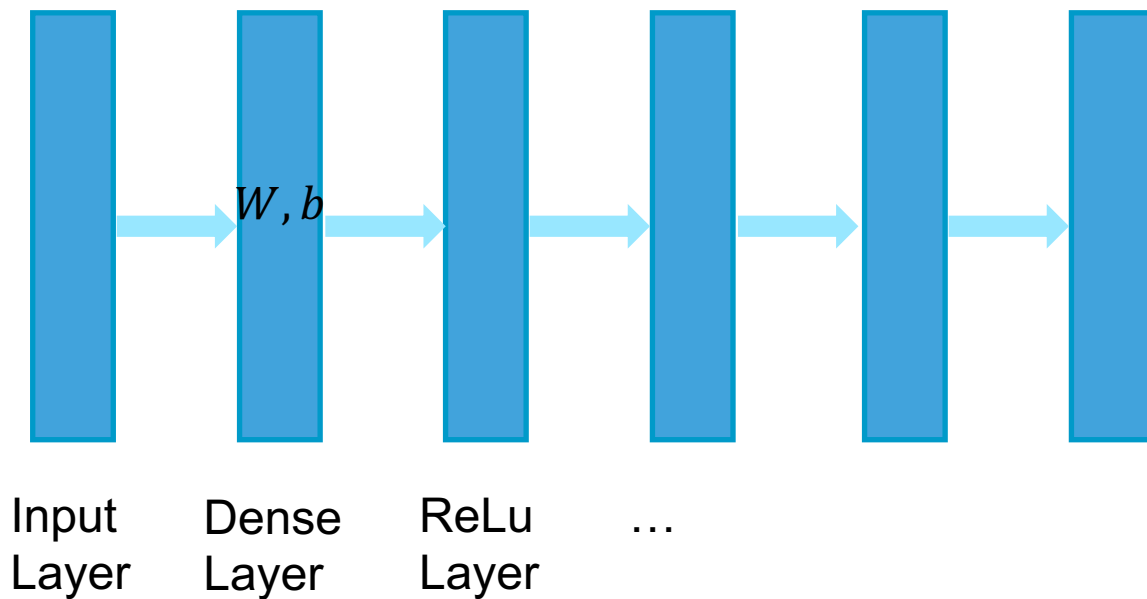$$y = Wx + b \qquad W \text{ and } b \text{ are the parameters to learn}$$

Activation Layer: Normally is a non-parametric layer. In the case of ReLu:

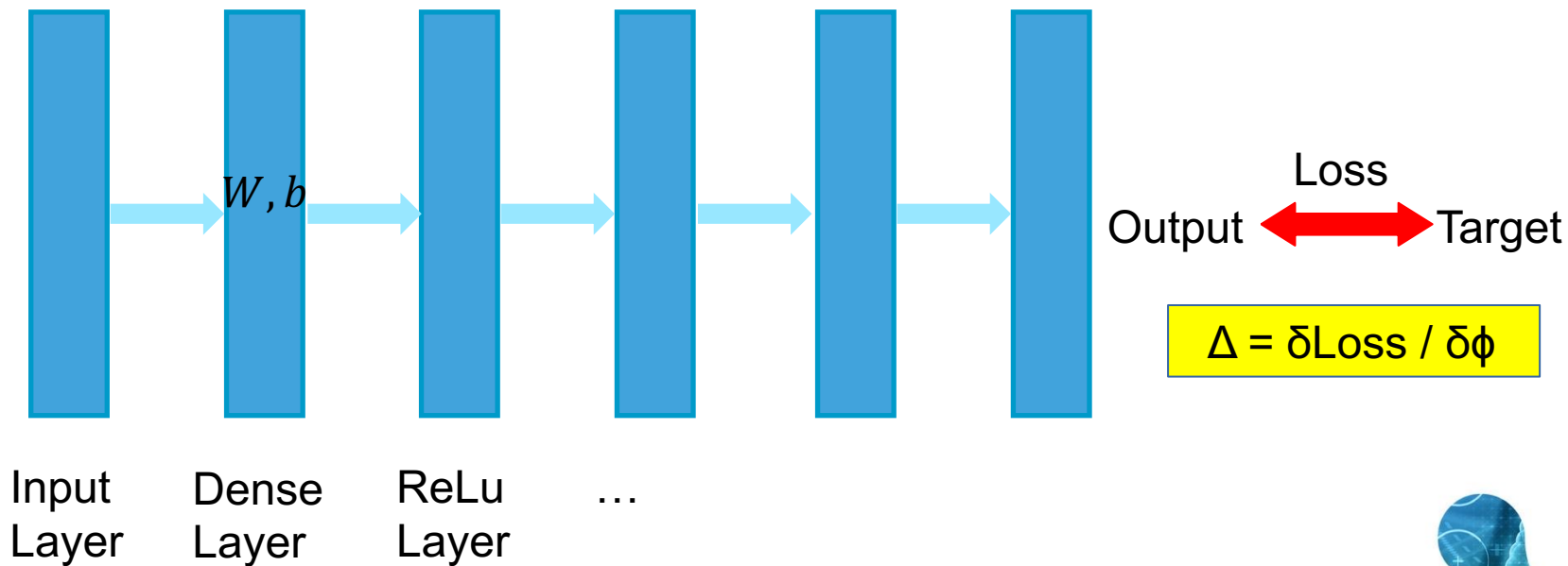$$y = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

What is a Neural Network?

$$W, b$$

Input
Layer

Dense
Layer

ReLu
Layer

…

What is a Neural Network?

W, b

Loss

Output ⟷ Target

$$\Delta = \delta Loss / \delta \phi$$

Input
Layer

Dense
Layer

ReLu
Layer

…

What is a Neural Network?

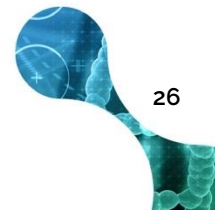Classification problems. n clases output with n neurons

Output: Softmax
$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad y_i = p(c = i | \boldsymbol{x})$$

Loss: Categorical cross-entropy
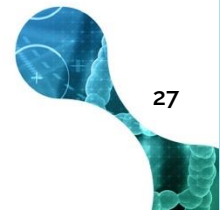$$\text{Loss} = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i$$

target

What is a Neural Network?

Regresion problems. d dimensions output with d neurons

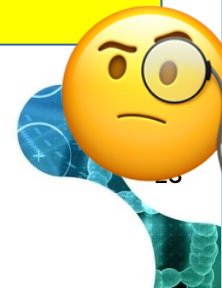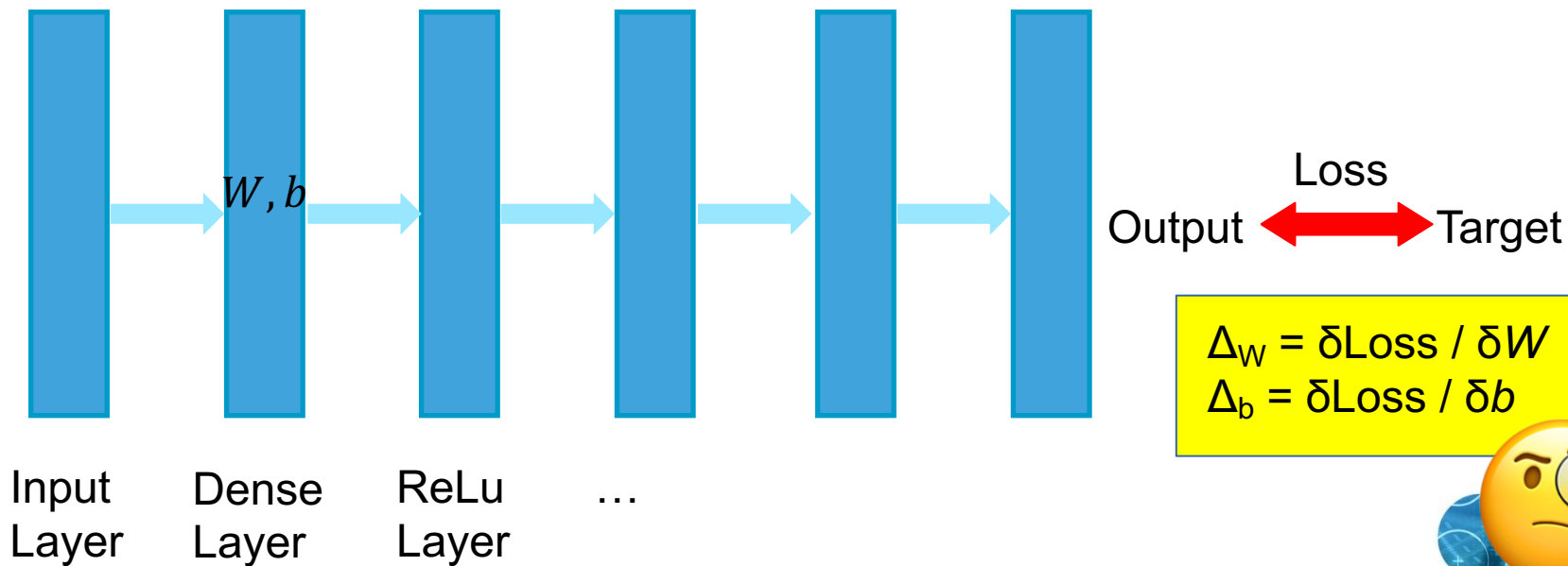Output: Linear $\qquad\qquad y_i = x_i$

Loss: Sum of
Quadratic Errors $\qquad Loss = \sum_{i=1}^{d}(y_i - \hat{y}_i)^2$

What is a Neural Network?
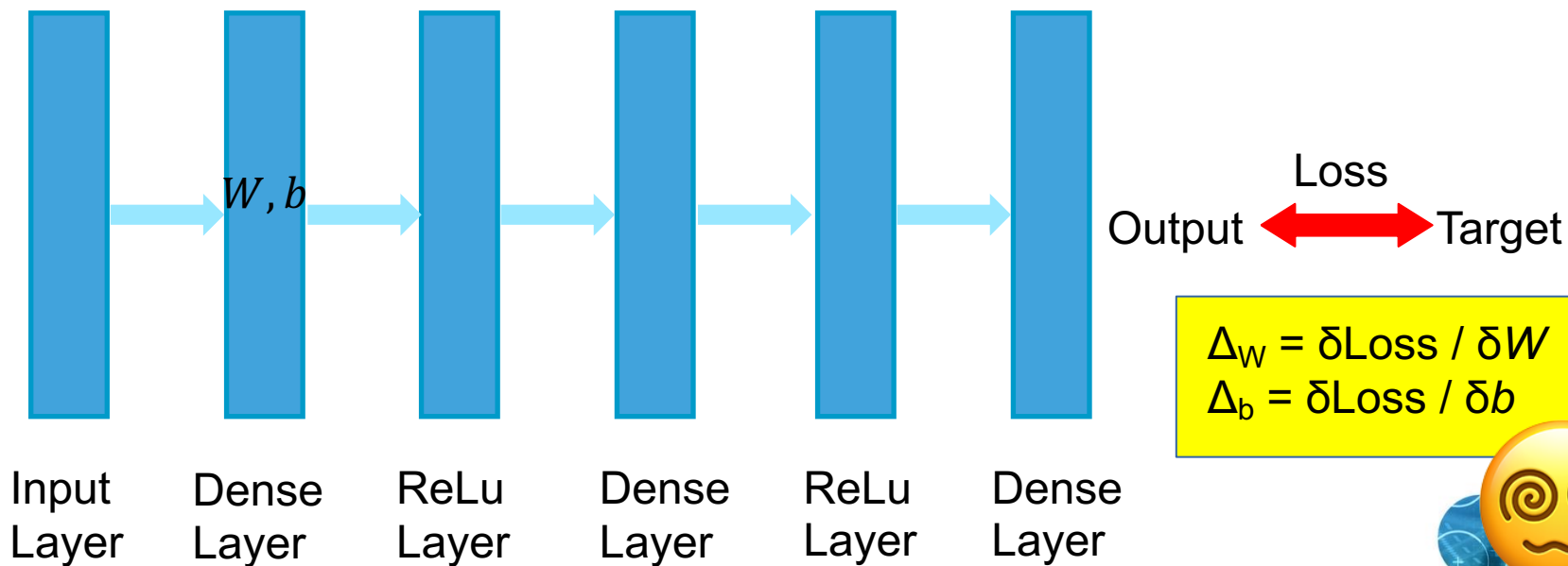
W, b

Loss

Output ⟷ Target

$\Delta_W = \delta Loss / \delta W$
$\Delta_b = \delta Loss / \delta b$

Input Layer   Dense Layer   ReLu Layer   …

What is a Neural Network?

$$W, b$$

Loss

Output ⟷ Target

$$\Delta_W = \delta\text{Loss} / \delta W$$
$$\Delta_b = \delta\text{Loss} / \delta b$$

| Input Layer | Dense Layer | ReLu Layer | Dense Layer | ReLu Layer | Dense Layer |

What is a Neural Network?

What is a Neural Network?

In order to compute the gradient $\Delta = \delta Loss / \delta \phi$ w.r.t any parameter we can use **the chain rule** to back-propagate the loss

This leads us to the well-know Backpropagation algorithm

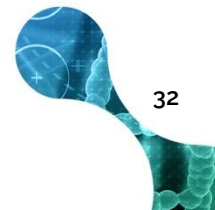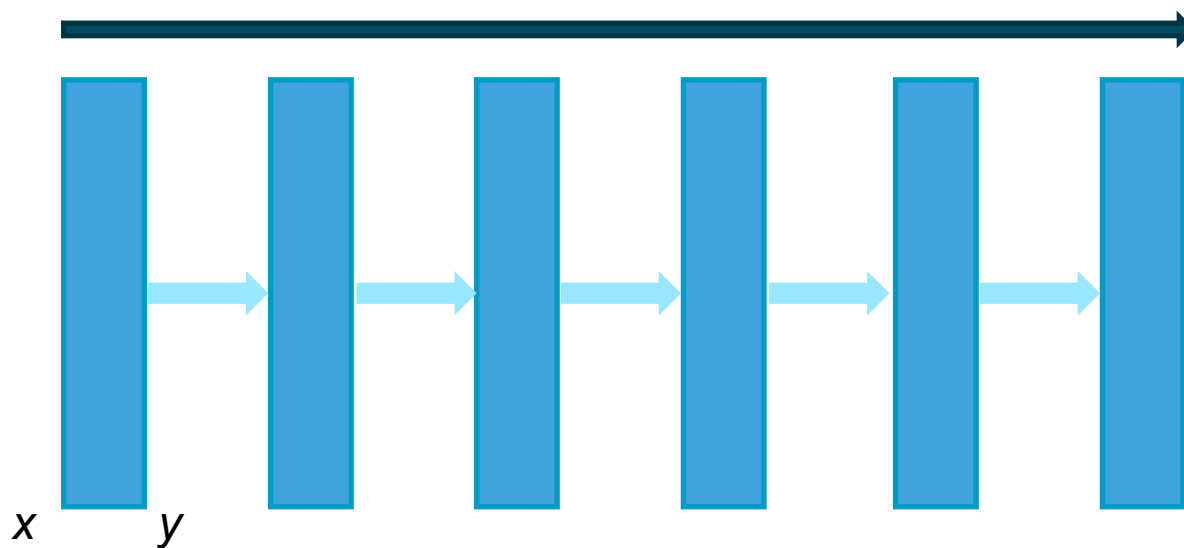Essentially the training phase of a NN is composed by three steps:

- Forward : the output of all the layers are computed
- Backward: the gradient w.r.t all the parameters of all the layers is computed (backpropagation)
- Update: the parameters are updated in the opposite direction of the gradient (optimizer)
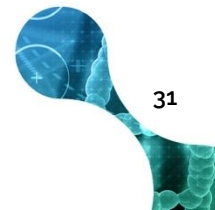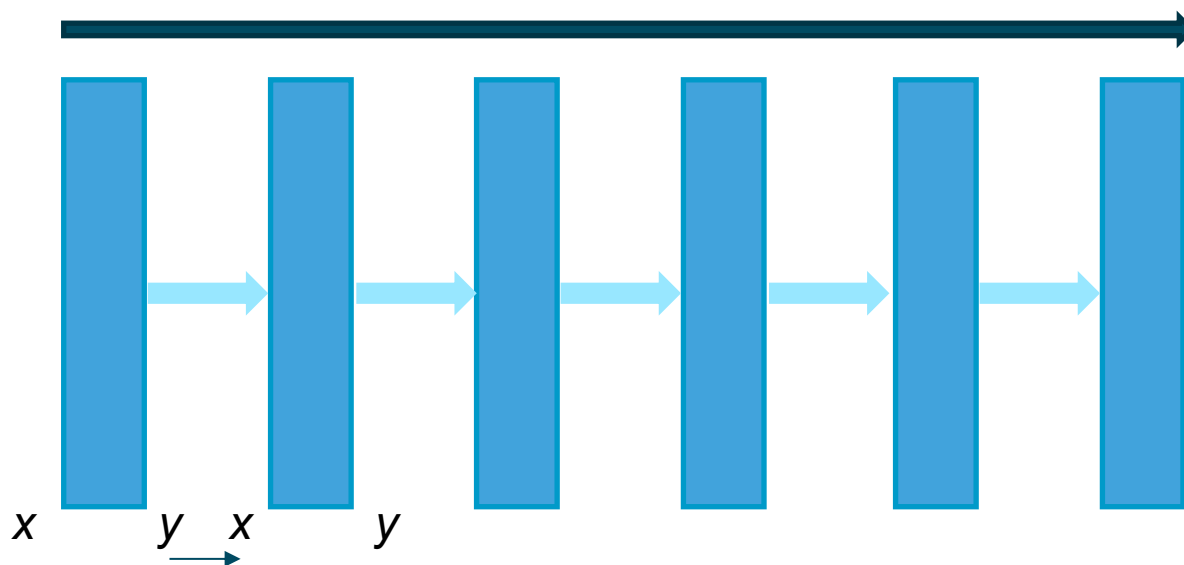
31

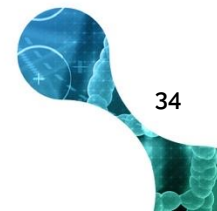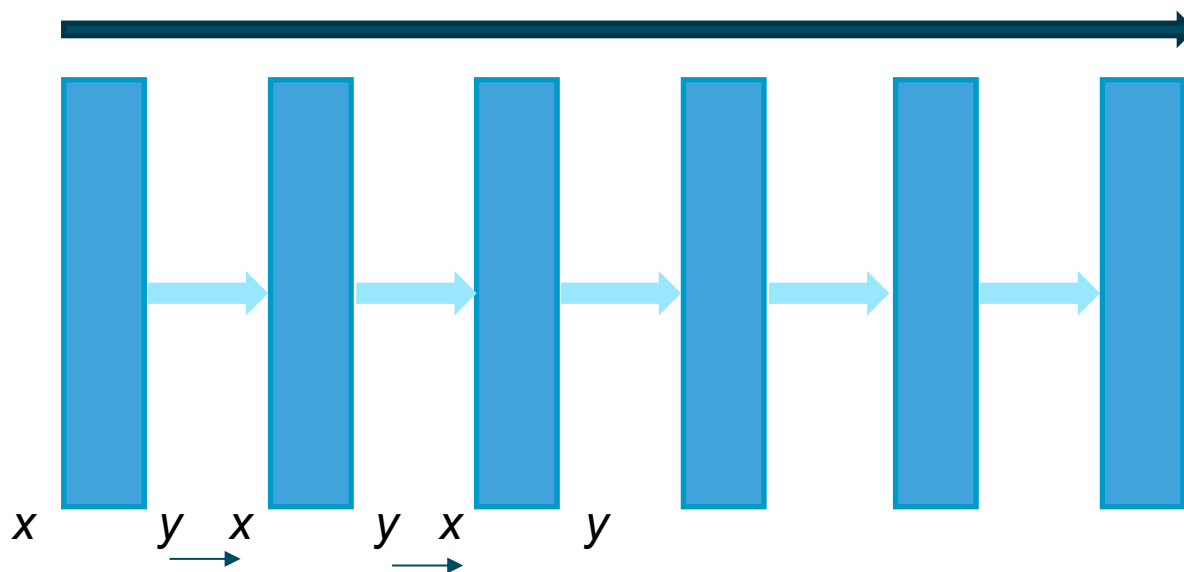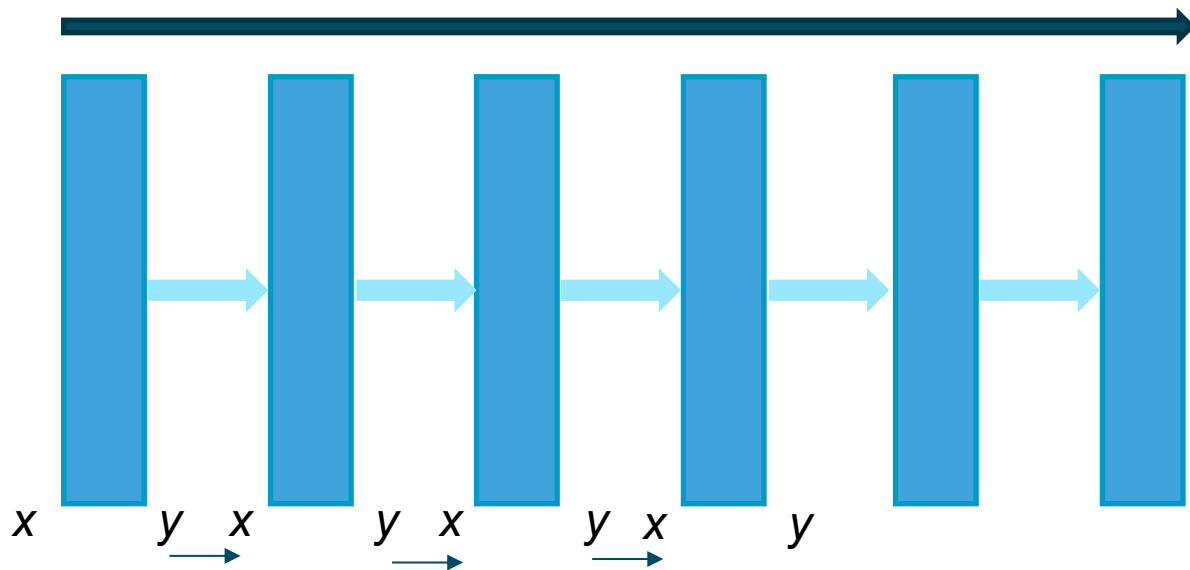# What is a Neural Network?

Forward

What is a Neural Network?

Forward

$x \qquad y \quad x \qquad y$

What is a Neural Network?

Forward

What is a Neural Network?

Forward

x y x y x y x y

What is a Neural Network?

Forward

x $\quad$ y $\quad$ x $\quad$ y $\quad$ x $\quad$ y $\quad$ x $\quad$ y $\quad$ x $\quad$ y

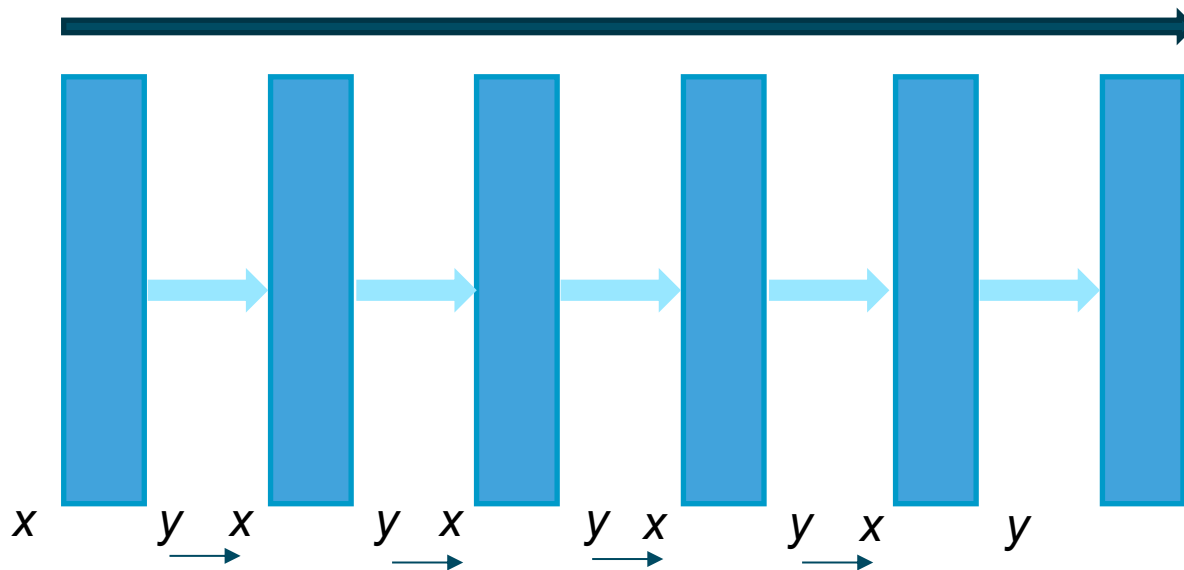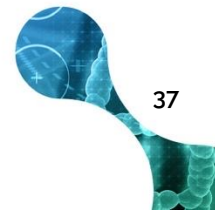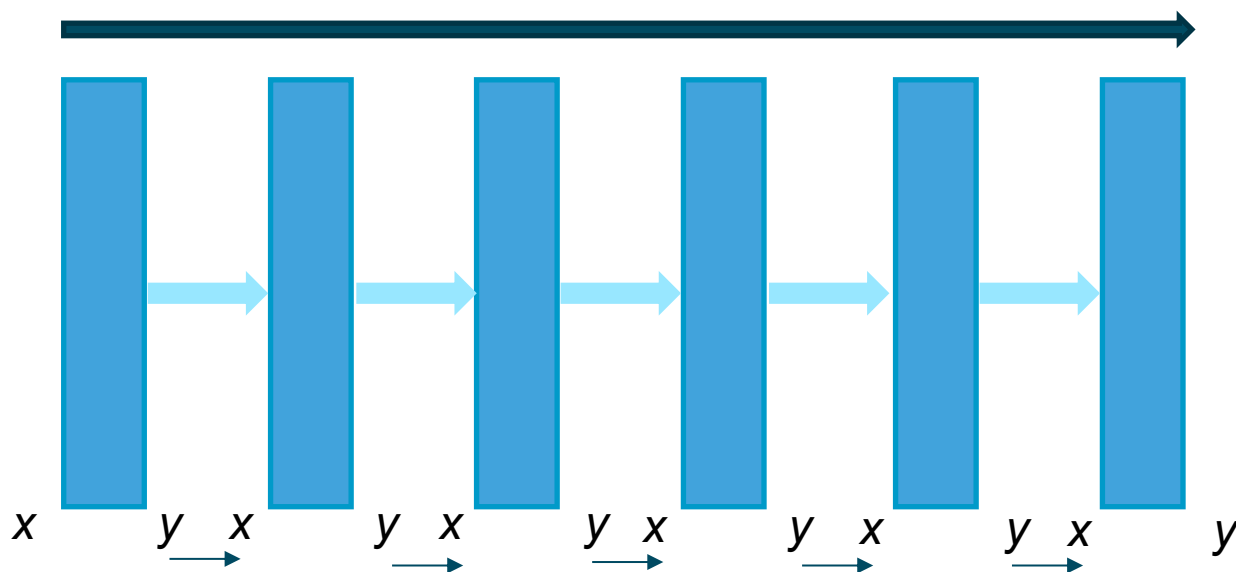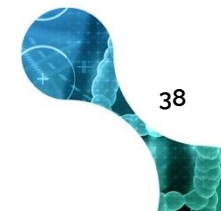# What is a Neural Network?

Forward

What is a Neural Network?

Forward
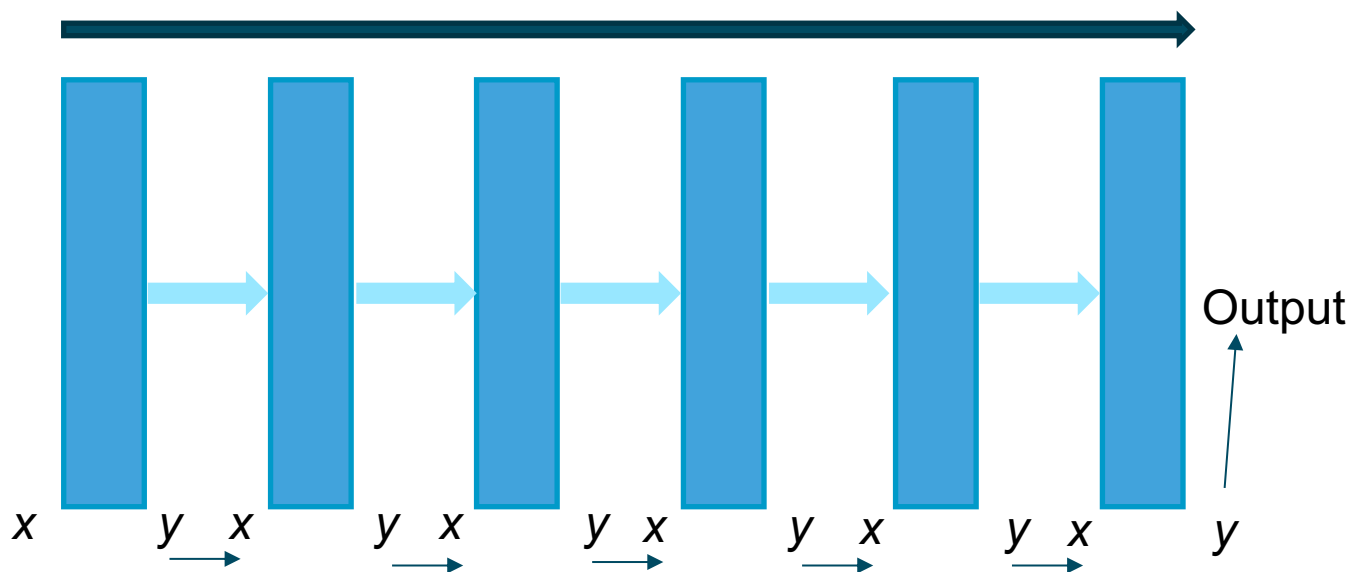

x y x y x y x y x y x y

Output

What is a Neural Network?

Forward

Input

Output

x y x y x y x y x y x y

What is a Neural Network?

Forward

Input

Output

Target

x y x y x y x y x y x y

What is a Neural Network?

Forward

Input

x y x y x y x y x y x y

Loss

Output Target

$$\Delta = \delta Loss / \delta \phi$$

What is a Neural Network?

Loss

Output ⟷ Target

$$\Delta = \delta Loss / \delta\phi$$

Backward

What is a Neural Network?

$$\Delta = \delta Loss / \delta \phi$$

Loss

Output ⟷ Target

Backward

What is a Neural Network?

$$\Delta = \delta Loss / \delta\phi$$

Loss

Output ⬅➡ Target

Backward

What is a Neural Network?

Loss

Output ⟷ Target

$$\Delta = \delta Loss / \delta \phi$$

Backward

What is a Neural Network?

$$\Delta = \delta Loss / \delta \phi$$

Loss

Output ⟷ Target

Backward

What is a Neural Network?

$$\Delta = \delta Loss / \delta \phi$$

Loss

Output ⟷ Target

Backward

What is a Neural Network?

| Input Layer | Dense Layer | ReLu Layer | Dense Layer | ReLu Layer | Dense Layer |
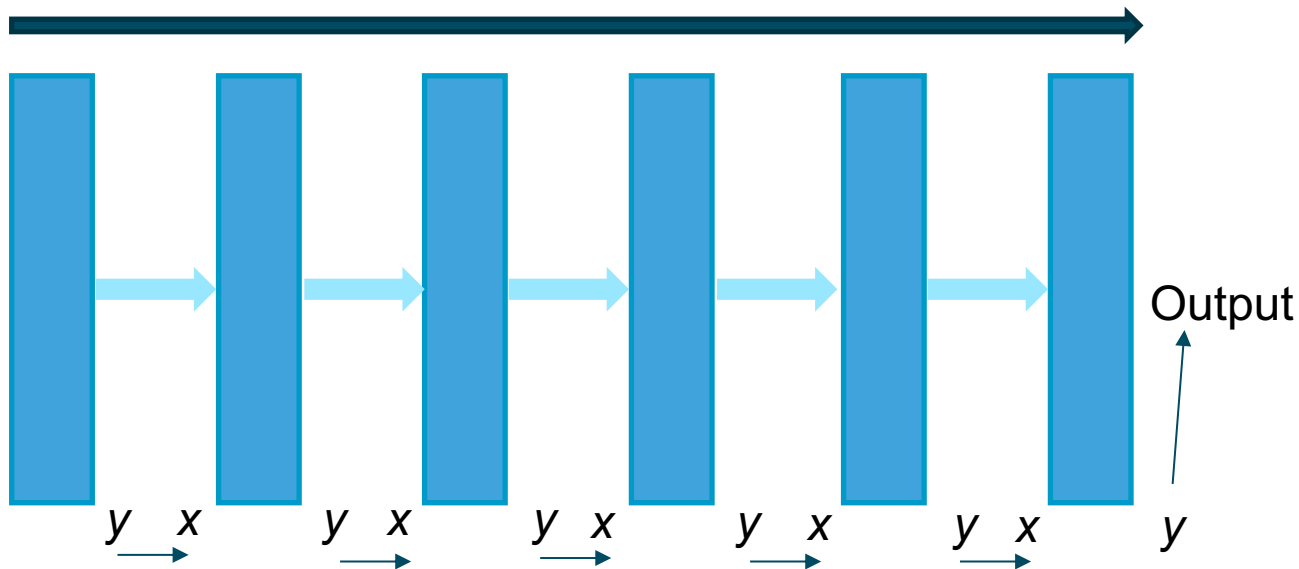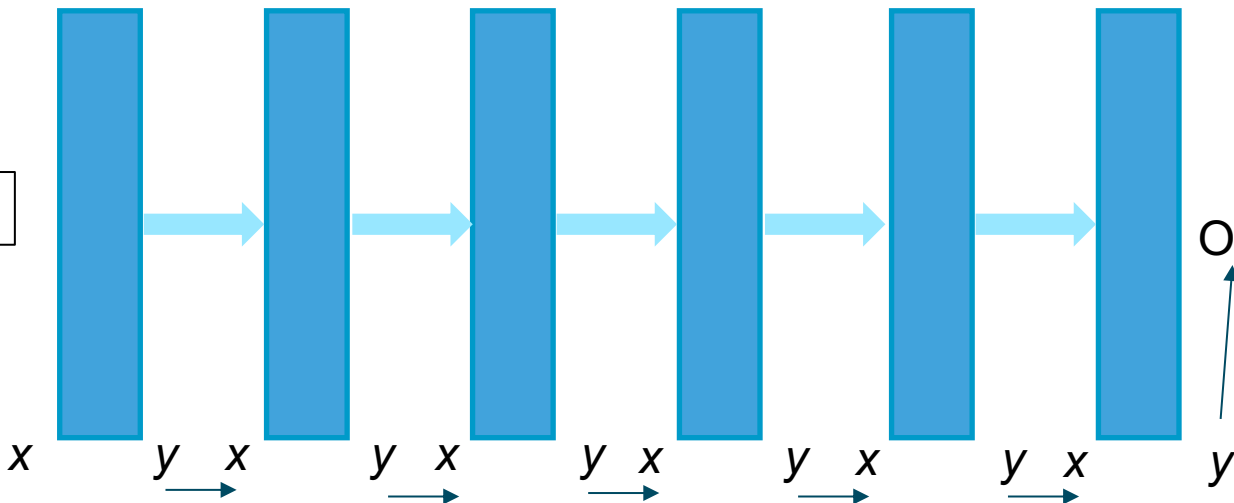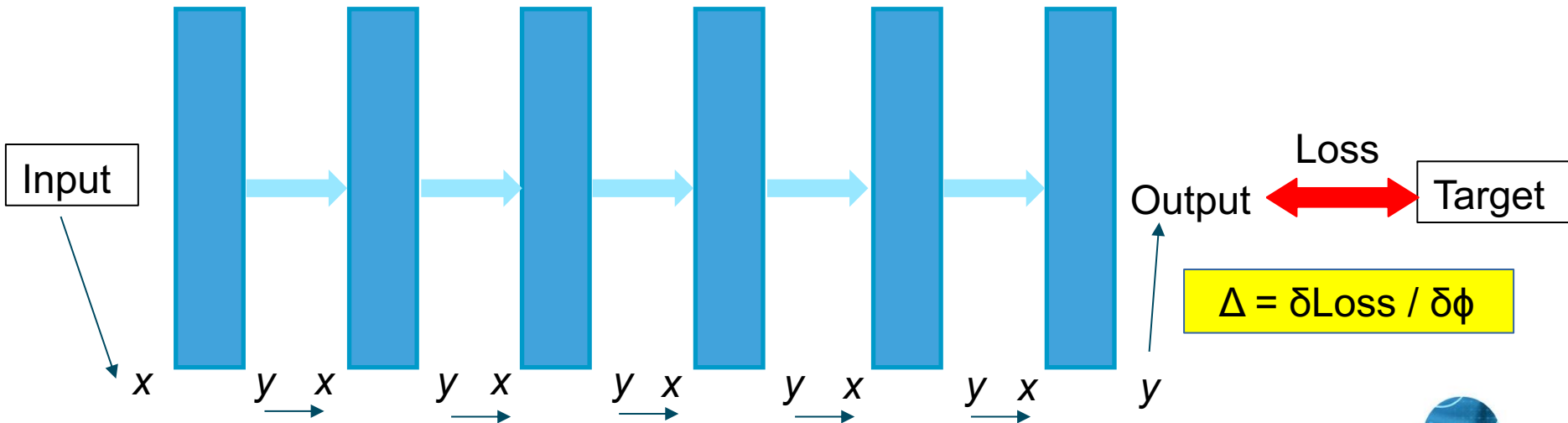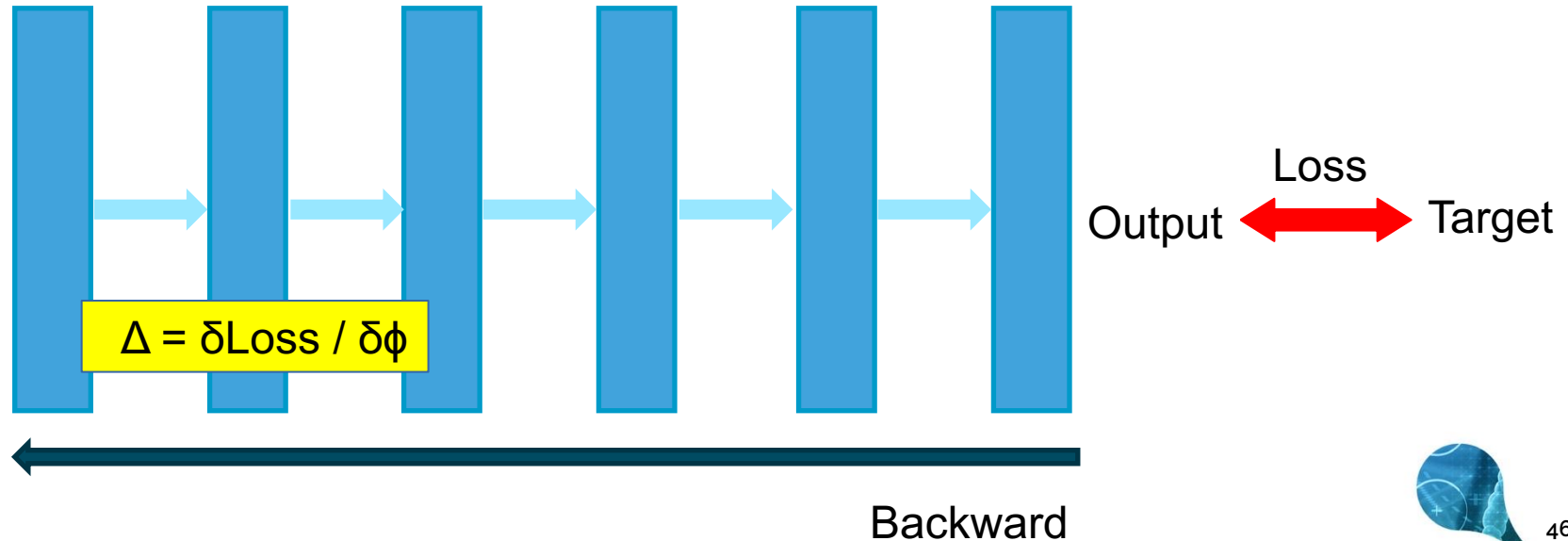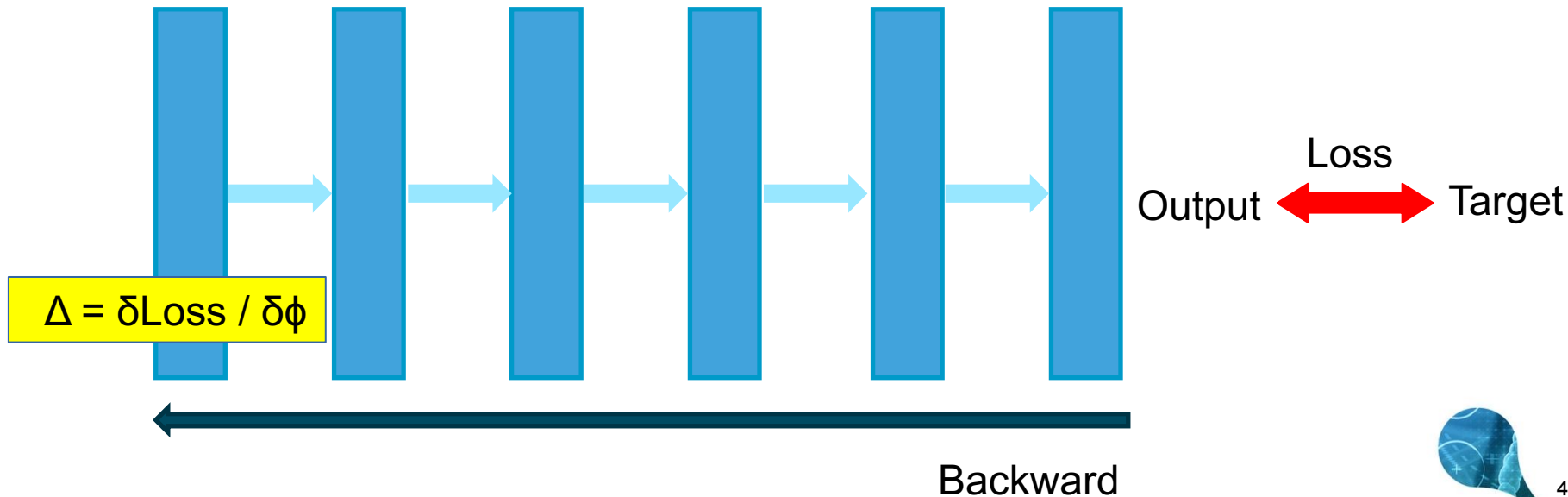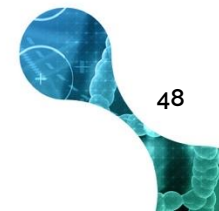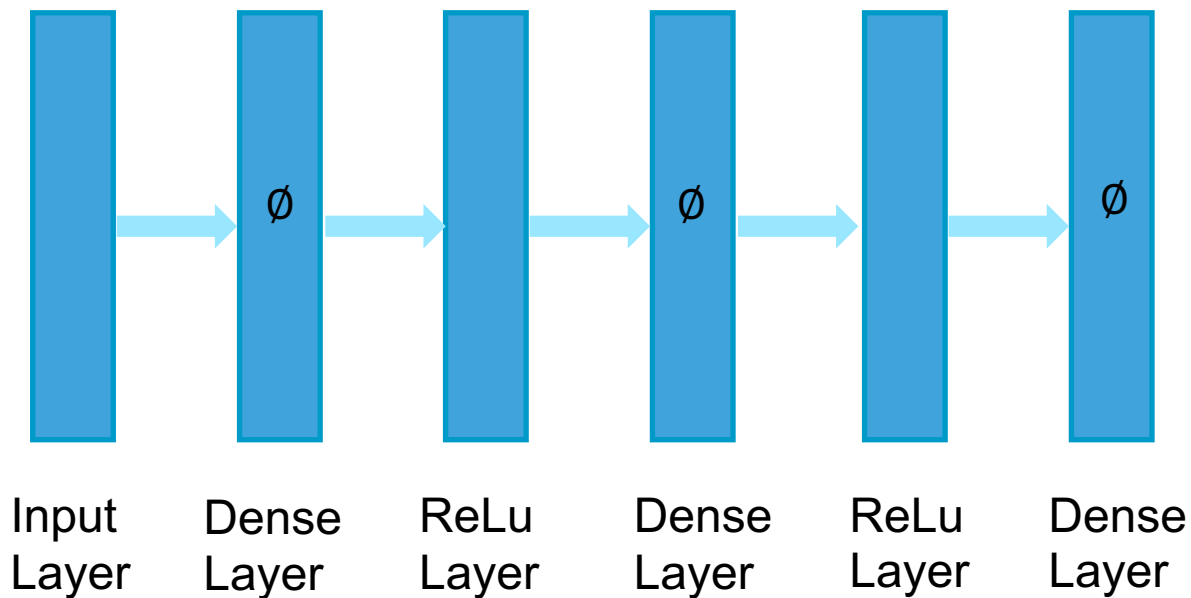
What is a Neural Network?

Update

$$W_1 = W_1 - \mu\Delta_{W1}$$
$$b_1 = b_1 - \mu\Delta_{b1}$$

$$W_2 = W_2 - \mu\Delta_{W2}$$
$$b_2 = b_2 - \mu\Delta_{b2}$$

$$W_3 = W_3 - \mu\Delta_{W3}$$
$$b_3 = b_3 - \mu\Delta_{b3}$$
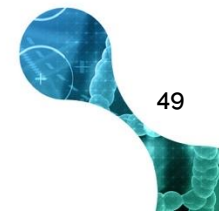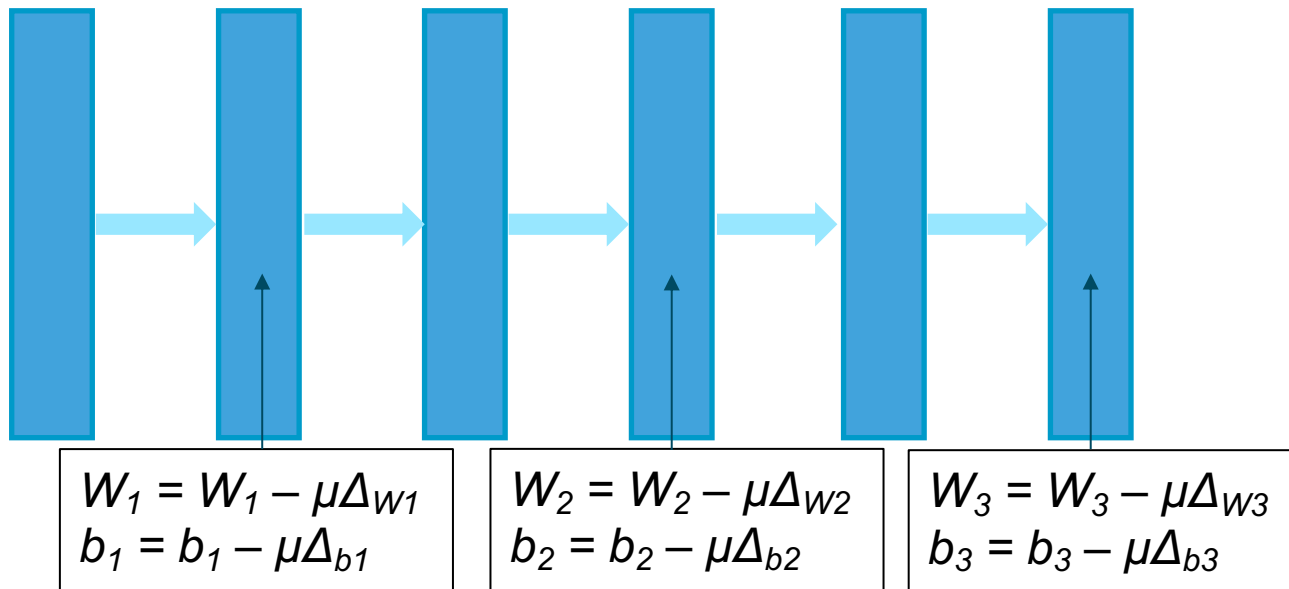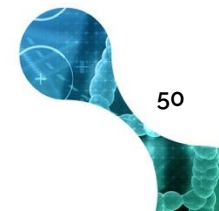
What is a Neural Network?

Training:
- Forward
- Backward
- Update

Inference:
- Forward

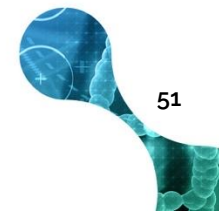# What is a Neural Network?

Training:
- Forward
- Backward
- Update

EDDL: model.fit(input, target)

Inference:
- Forward

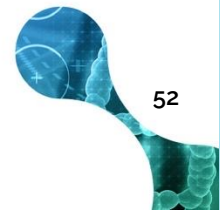EDDL: output = model.predict(input)

What is a Neural Network?

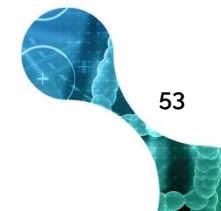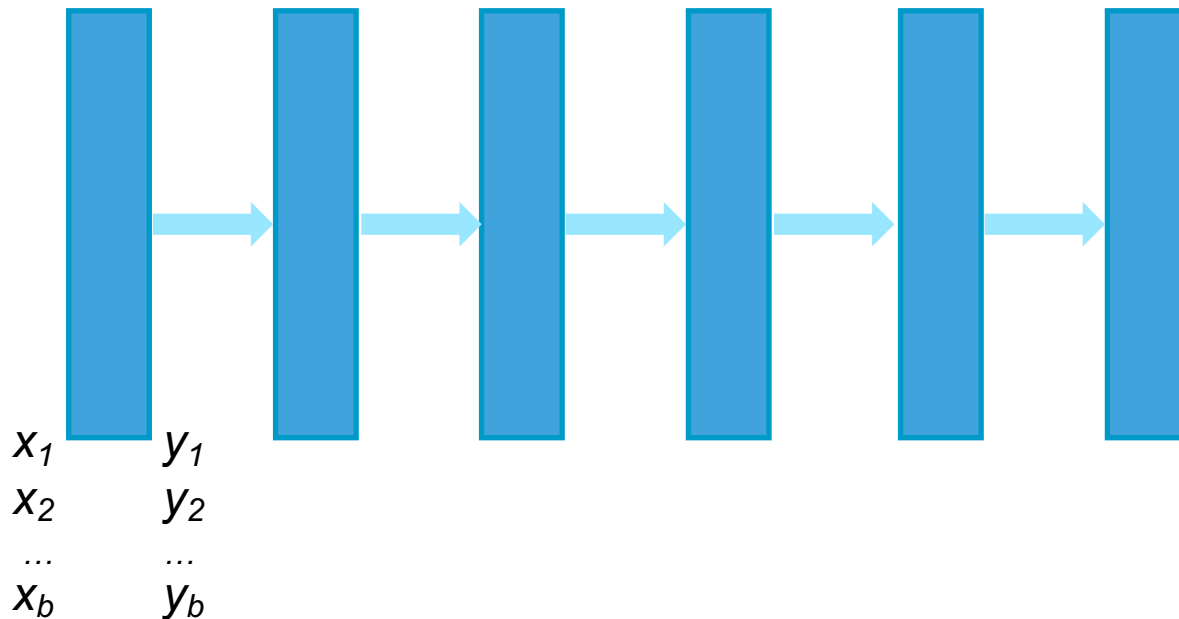Training:
- Forward
- Backward
- Update

The samples (input,target) are presented in batch, then the gradient is computed for all the samples in a **batch**

What is a Neural Network?

Forward

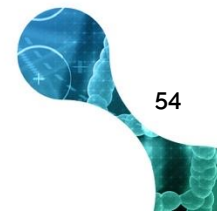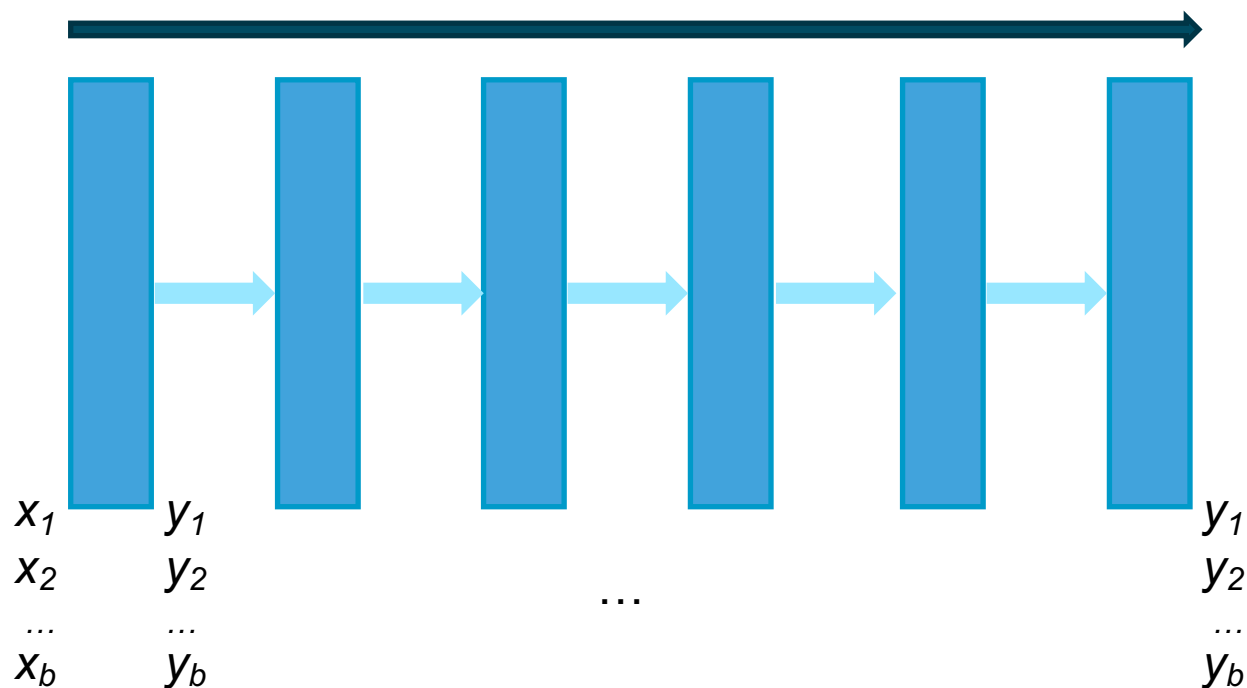x_1 y_1
x_2 y_2
…   …
x_b y_b

What is a Neural Network?

Forward

x_1 y_1 y_1
x_2 y_2 y_2
...   .
x_b y_b y_b

What is a Neural Network?

Forward

x_1
x_2
…
x_b

y_1
y_2
…
y_b

…

y_1
y_2
…
y_b

y_1
y_2
…
y_b

Loss

t_1
t_2
…
t_b

What is a Neural Network?

Forward

$$\Delta = \sum_b \frac{\partial Lossb}{\partial \emptyset}$$
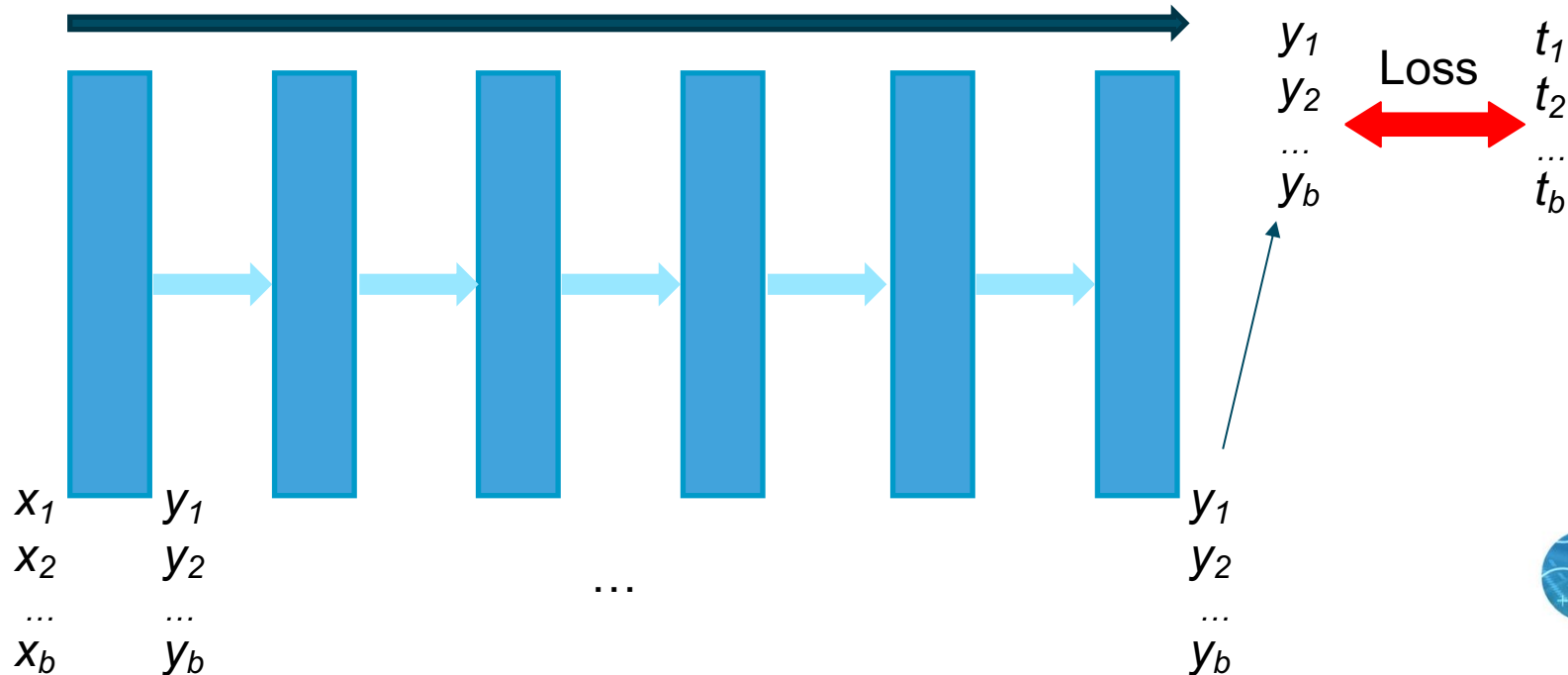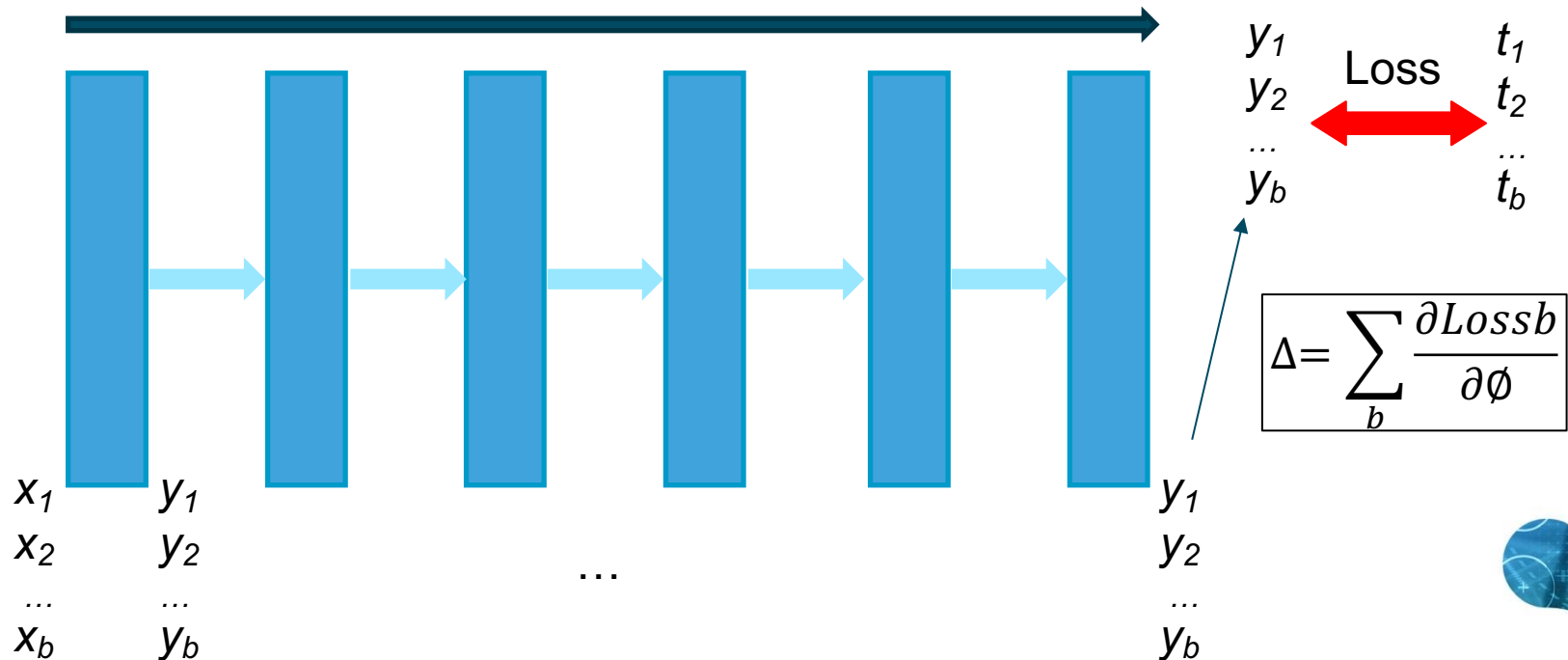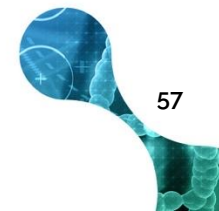
What is a Neural Network?

Training:
- Forward
- Backward
- Update

EDDL: model.fit(input, target, batch_size)

Inference:
- Forward

EDDL: output = model.predict(input)
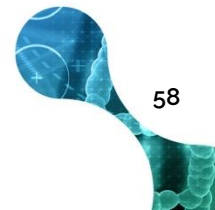
What is a Neural Network?

Training

Definition of **epoch**: train all the samples, then is to train the number of batches that fit in your training set.

Training set of 50.000 samples
Batch_size= 100

Epoch means to train 500 batches (randomly selected) covering all the samples of the training set

What is a Neural Network?

Training:
- Forward
- Backward
- Update

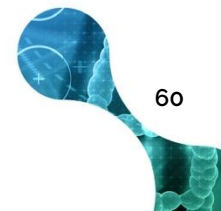EDDL: model.fit(input, target, batch_size, epochs)

Inference:
- Forward

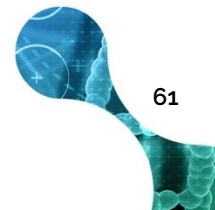EDDL: output = model.predict(input)

EDDL Basic

EDDL Basic

EDDL is an open source library for Distributed Deep Learning and Tensor Operations in C++ for **CPU**, **GPU** and **FPGA**. EDDL is developed inside the DeepHealth project.

EDDL provides a high level API to develop Deep learning projects in C++

**PyEDDL** is a Python wrapper for the EDDL

61

EDDL Basic

EDDL is like Keras in the way the models are defined and the training is carried out. The main difference between EDDL and other toolkits is the introduction of the Computing Service (CS) object.

The DL model created with EDDL has a CS attached where this model will be deployed. This CS could be CPU, GPU or FPGA and even we could define a distributed CS.

This CS definition only affects to a single line in your programs.
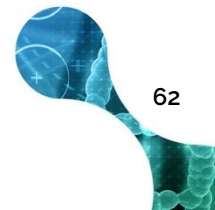
EDDL Basic

```
download_mnist();

// Define network
layer in = Input({784});
layer l = in;

l = LeakyReLu(Dense(l, 1024));
l = LeakyReLu(Dense(l, 1024));
l = LeakyReLu(Dense(l, 1024));

layer out = Softmax(Dense(l, 10), -1);
model net = Model({in}, {out});

// Computing service
compserv cs = CS_CPU();
// cs = CS_GPU({1},"low_mem"); // one GPU
// cs = CS_GPU({1,1},100); // two GPU
// cs = CS_FPGA({1});
```

```
// Build model
build(net,
    adam(0.001), // Optimizer
    {"softmax_cross_entropy"}, // Losses
    {"categorical_accuracy"}, // Metrics
    cs ); // Computing service

// Train model
fit(net, {x_train}, {y_train}, 32, 10);

// Evaluate
evaluate(net, {x_test}, {y_test});
```
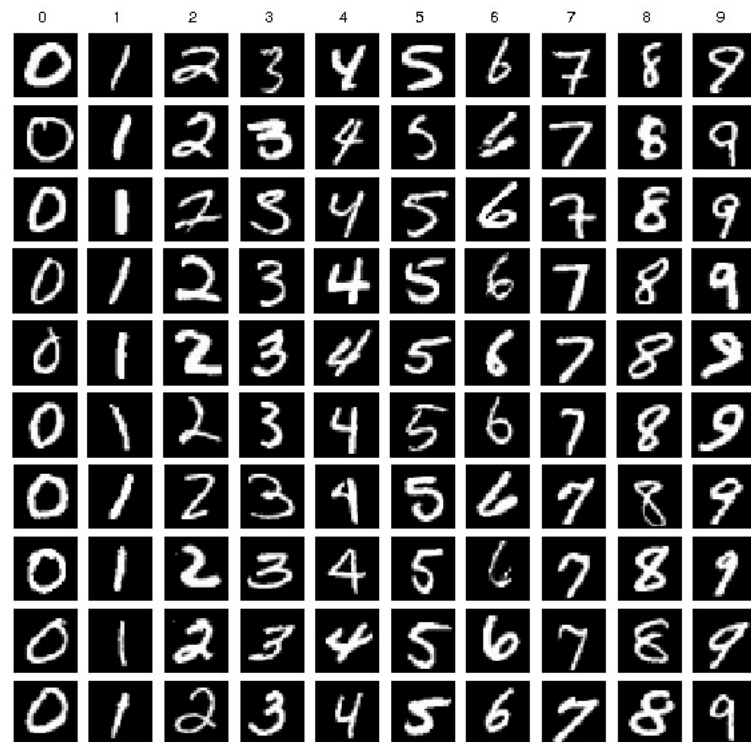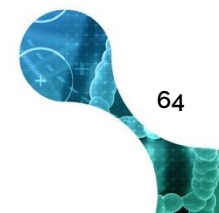
63

EDDL Basic

MNIST Digits Classification


28x28 images (2D)

stored as 784 pixels (1D)

10 classes

EDDL Basic   Data

```python
# Download dataset
eddl.download_mnist()
# Load the dataset
x_train = Tensor.load('mnist_trX.bin')
y_train = Tensor.load('mnist_trY.bin')
x_test = Tensor.load('mnist_tsX.bin')
y_test = Tensor.load('mnist_tsY.bin')

# Preprocess the images. From [0-255] to [0-1]
x_train.div_(255.0)
x_test.div_(255.0)

# Show data shape
print('Dataset shape:')
print(f'Train split images: {x_train.shape}')
print(f'Train split labels: {y_train.shape}')
print(f'Test split images: {x_test.shape}')
print(f'Test split labels: {y_test.shape}')
```
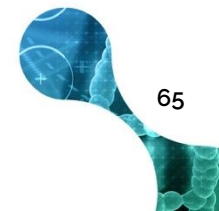
Model

```python
# Define the model topology
num_classes = 10
in_ = Input([784])
layer = Reshape(in_, [1, 28, 28])   # EDDL needs channel first images
layer = ReLu(BatchNormalization(Conv(layer, 32, [3, 3]), affine=True))
layer = MaxPool(layer, [2, 2])
layer = ReLu(BatchNormalization(Conv(layer, 64, [3, 3]), affine=True))
layer = MaxPool(layer, [2, 2])
layer = ReLu(BatchNormalization(Conv(layer, 128, [3, 3]), affine=True))
layer = MaxPool(layer, [2, 2])
layer = ReLu(BatchNormalization(Conv(layer, 256, [3, 3]), affine=True))
layer = GlobalAveragePool(layer)
layer = Flatten(layer)
out_ = Softmax(Dense(layer, num_classes))

# Create the model
model = eddl.Model([in_], [out_])
```

66
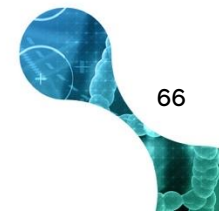
EDDL Basic   Learning

```python
# Build the model to prepare it for training or inference
eddl.build(model,
           opt,                              # Optimizer
           ['categorical_cross_entropy'],    # Losses
           ['accuracy'],                     # Metrics
           cs)                               # Computing Service


# Show the model layers
eddl.summary(model)


eddl.fit(model, [x_train], [y_train], batch_size, epochs)   # Train

eddl.evaluate(model, [x_test], [y_test], args.batch_size)   # Validation
```
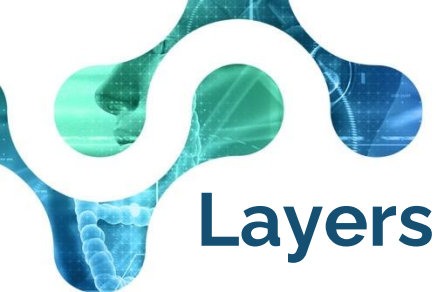
Layers and Tensors
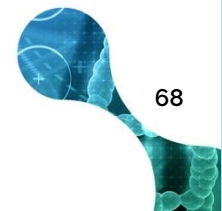
Layers

Dense

- **Input a 1D Tensor (d)**
- **Output a 1D Tensor (d')**

- **Operation:** $y = Wx + b$

- **Parameters: W,b**
  - **W is a (d',d) 2D Tensor**
  - **B is a (d') 1D Tensor**
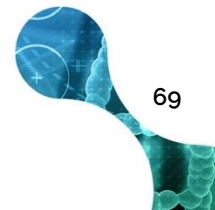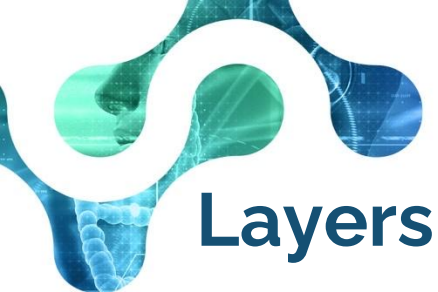
Layers

Activation

- **Input any D Tensor**
- **Output the same dim as input Tensor**

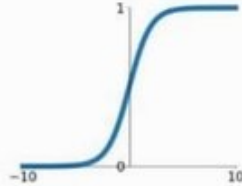- **Operation: depends on activation function**
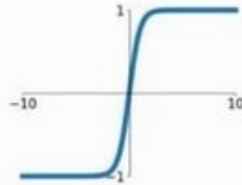
- **Parameters: NO**
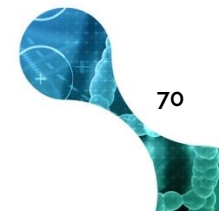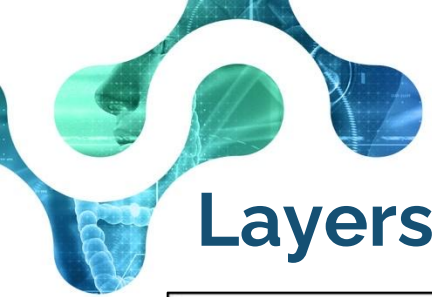
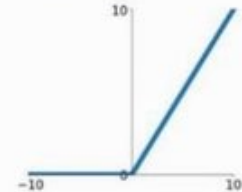# Layers

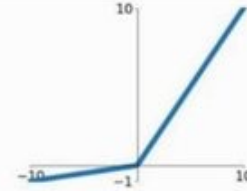

Sigmoid
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh
$$\tanh(x)$$

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

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

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

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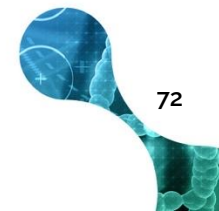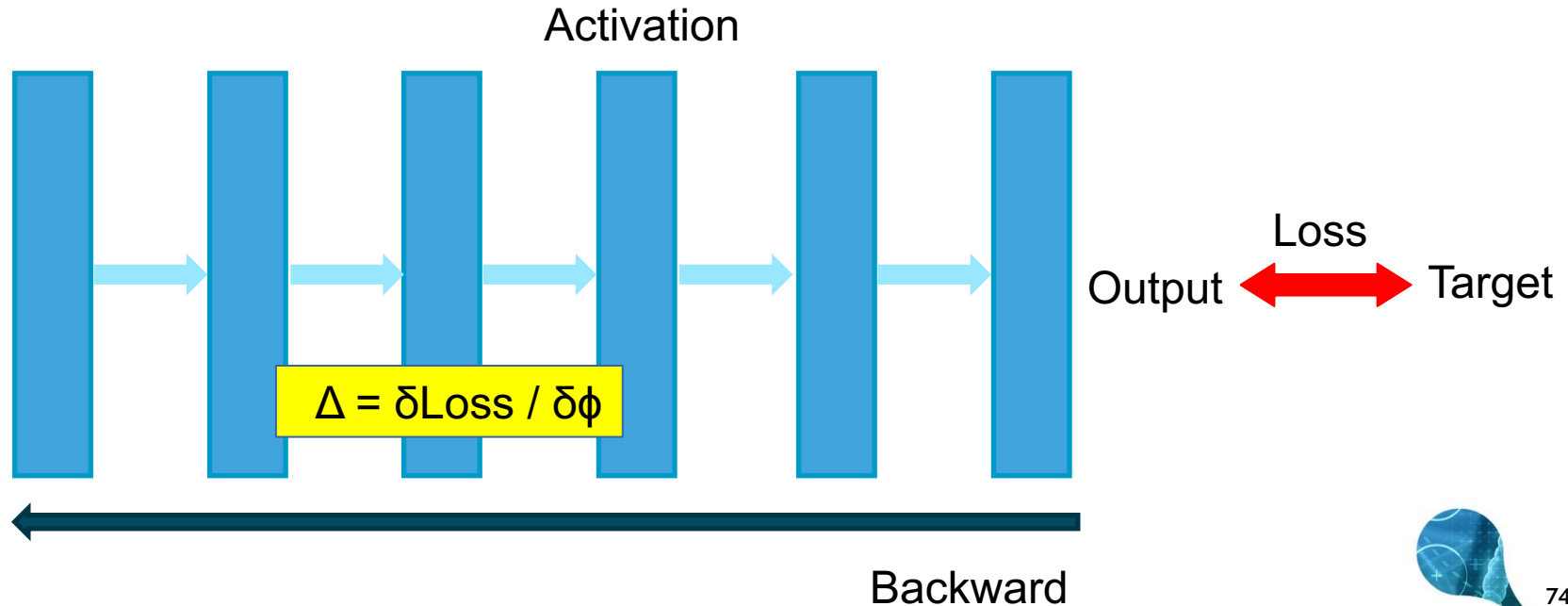$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$
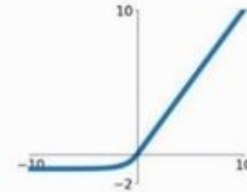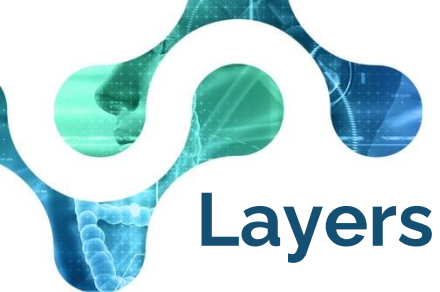
Layers

Activation

- **The backpropagated gradient is multiply by the derivative of the activation function**

What is a Neural Network?
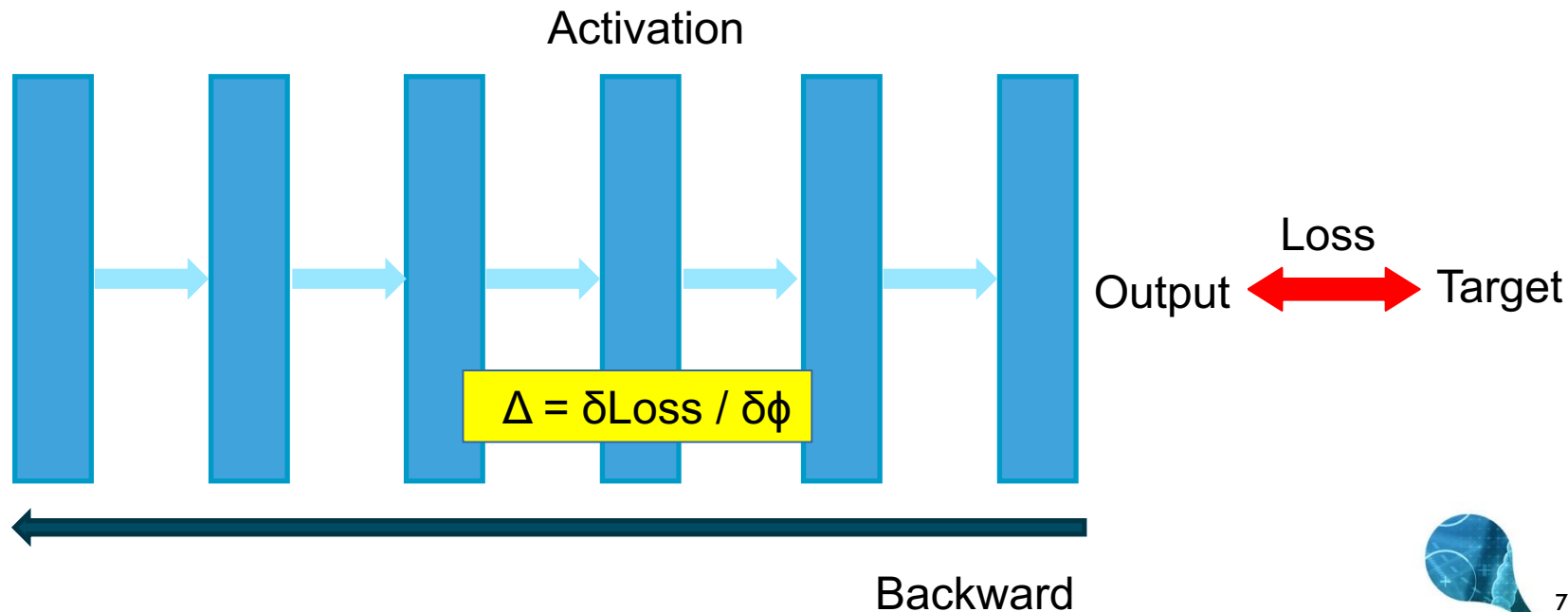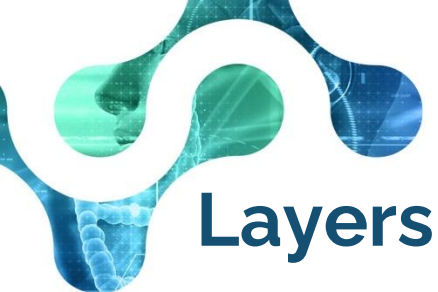
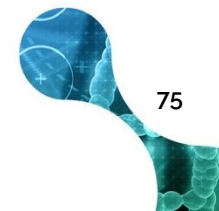Activation

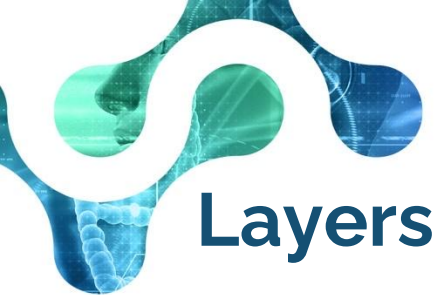$$\Delta = \delta Loss\ /\ \delta\phi$$

Loss

Output ⟷ Target

Backward

What is a Neural Network?

Activation

$$\Delta = \delta Loss / \delta\phi$$

Loss

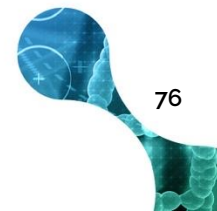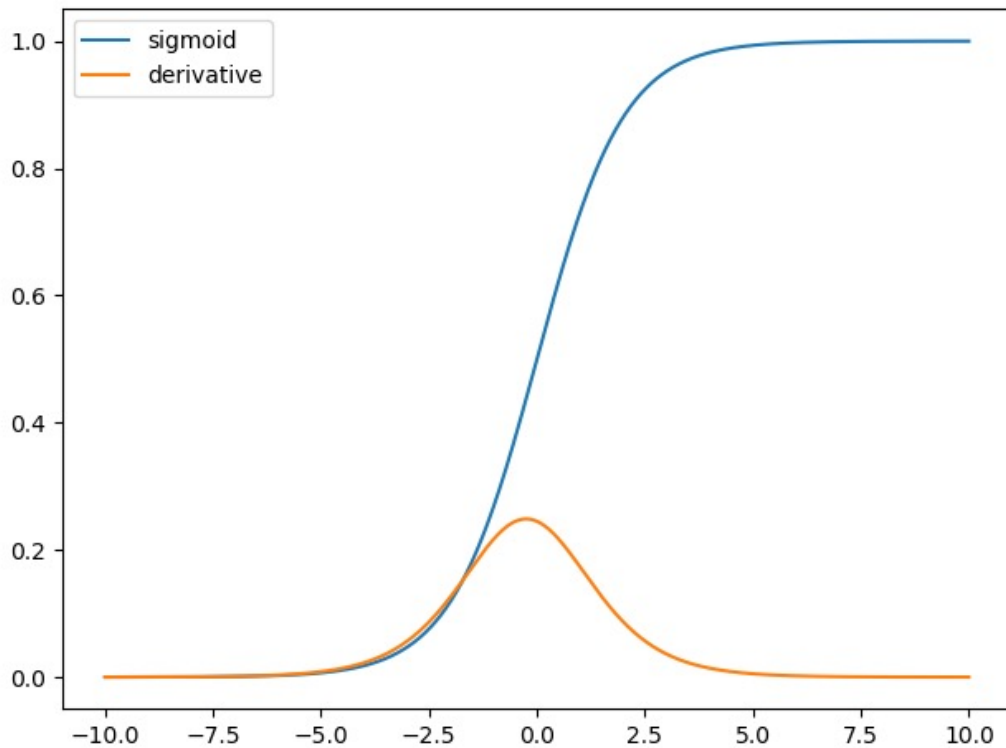Output ⟷ Target

Backward

Layers

Activation

- **The backpropagated gradient is multiply by the derivative of the activation function**

- **Vanishing gradient problem with some activation functions**
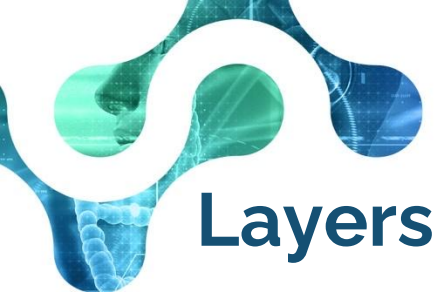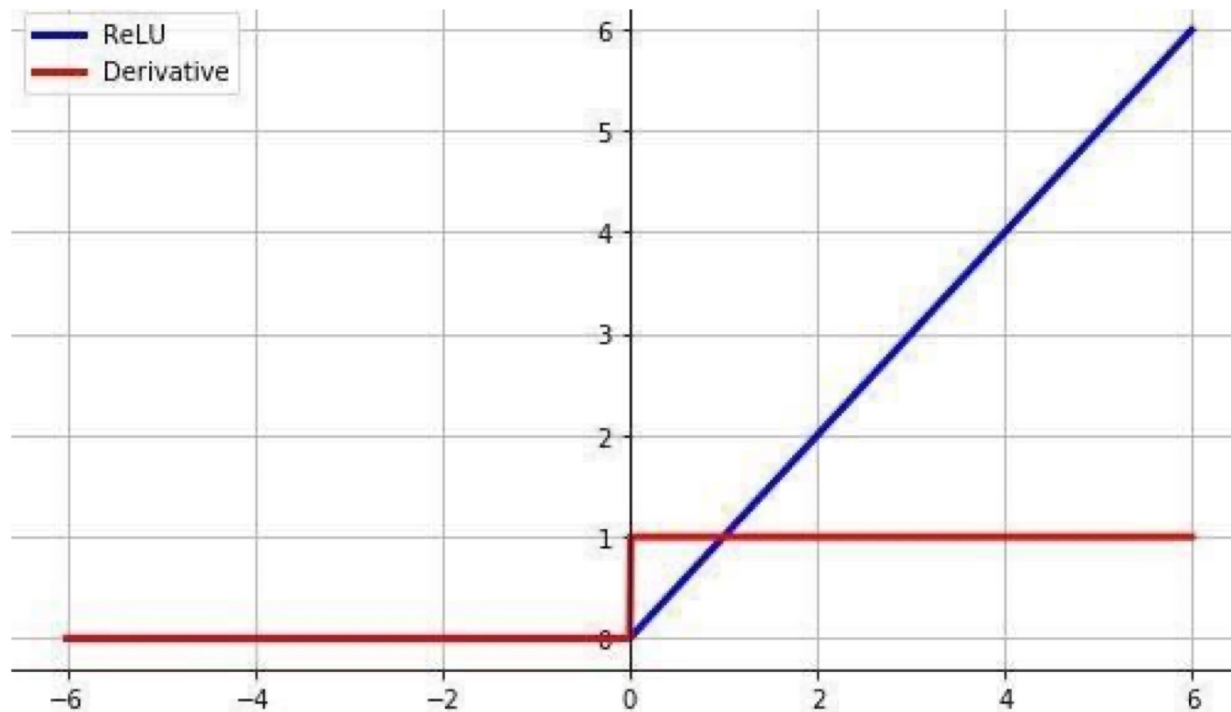
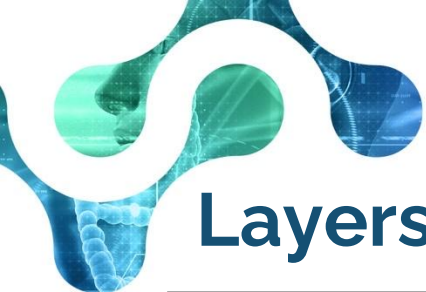- **Most used: ReLu, ELU and similar**

Layers

Activation

Layers
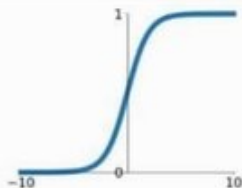
Activation

Layers

Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$		**Leaky ReLU** $\max(0.1x, x)$	
tanh $\tanh(x)$		**Maxout** $\max(w_1^T x + b_1, w_2^T x + b_2)$	
ReLU $\max(0, x)$		**ELU** $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$	

Layers

Activation - Softmax

Output $\qquad y_i \quad = \dfrac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad y_i = p(c = i|\boldsymbol{x})$
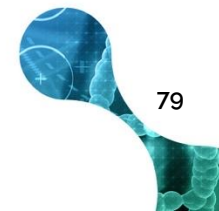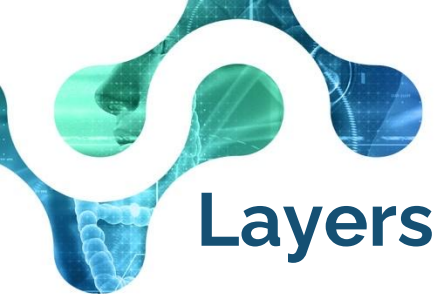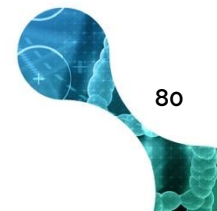
Layers

More Layers?

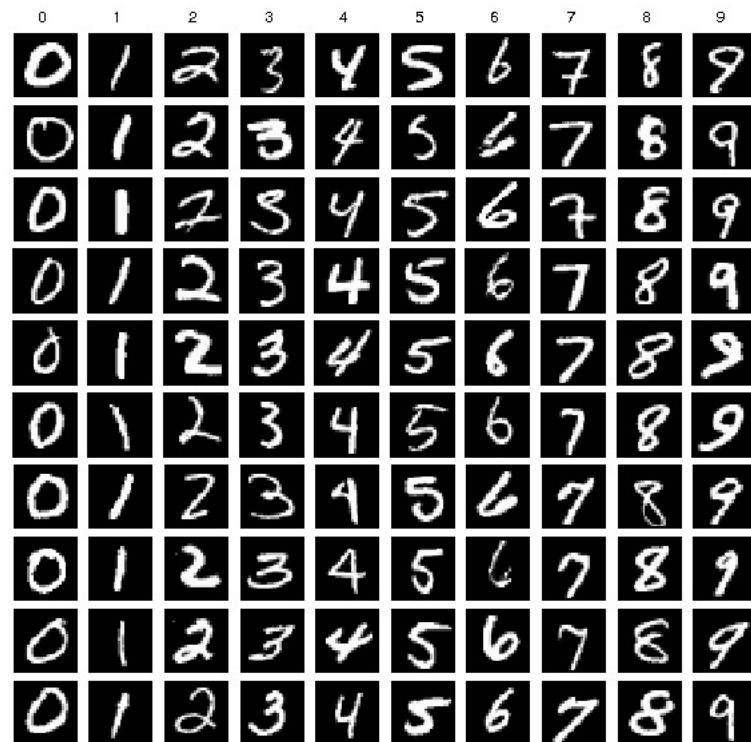[Deep Learning features](#)

EDDL with Dense topology (MLP)

28x28 images (2D)

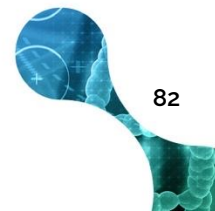stored as 784 pixels (1D)
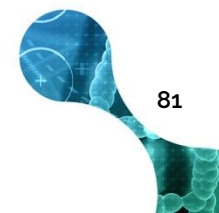
10 classes

EDDL with Dense topology (MLP)

```
num_classes = 10

in  = eddl.Input([784])

layer = eddl.ReLu(eddl.Dense(in, 1024))
layer = eddl.ReLu(eddl.Dense(layer, 512))
out = eddl.Softmax(eddl.Dense(layer, num_classes))

net = eddl.Model([in], [out])
```

EDDL with Dense topology (MLP)
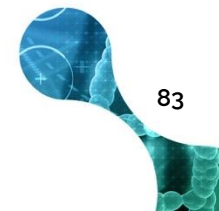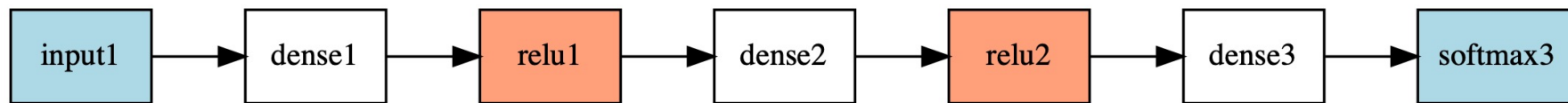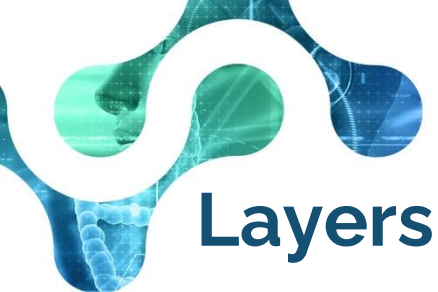
input1 → dense1 → relu1 → dense2 → relu2 → dense3 → softmax3

Layers

Conv2D

- **Input a 3D Tensor (channels_in, height_in, width_in)**
- **Output a 3D Tensor (channels_out, height_in, width_out)**

- **Operation:** $y = K * x + b$

- **Parameters: K,b**
  - **K is a (num_filters, channels_in, kernel_height, kernel_width) 4D Tensor**
  - **B is a (num_filters) 1D Tensor**

Layers

INPUT MAP

M

Ir

Ic

KERNELS

M

Kc

Kr

...

M

N

OUTPUT MAP

N

Or

Oc

Layers

Conv2D

kernel

image height

image depth

image width
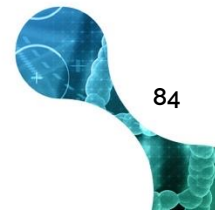
Layers

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 \quad + \quad -498 \quad + \quad 164 \quad + 1 = -25$$

Bias = 1

Output

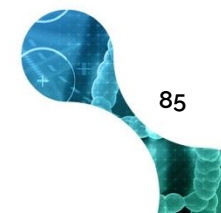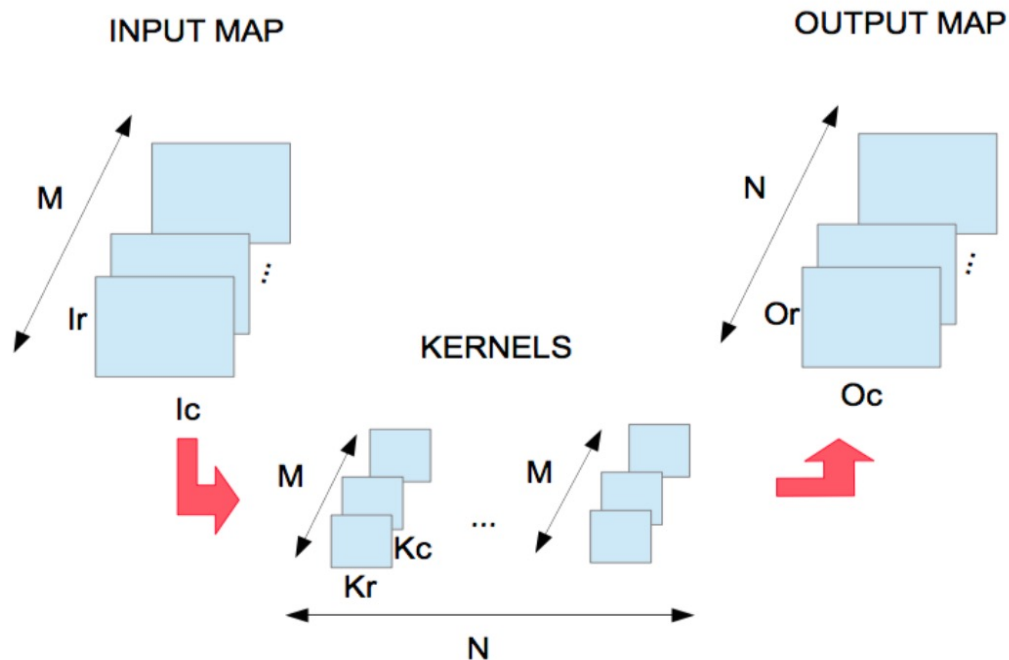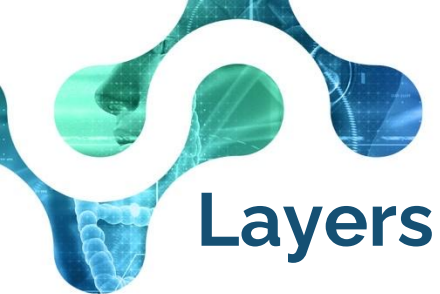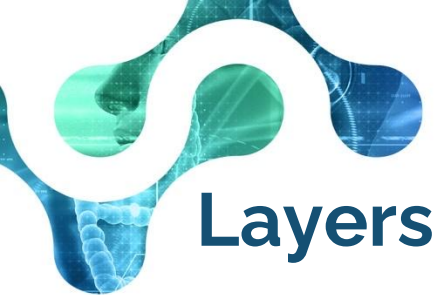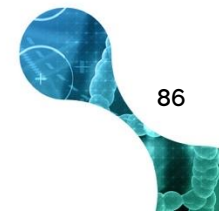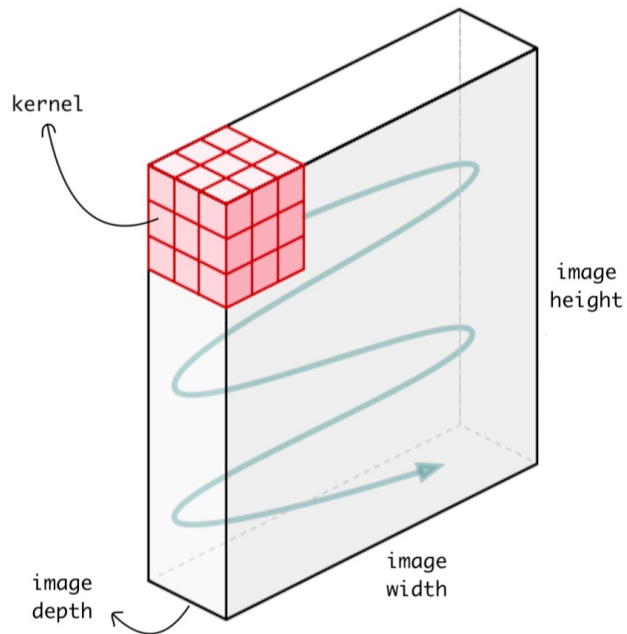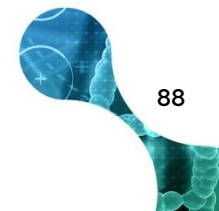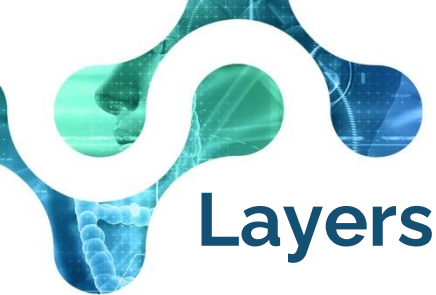| -25				...
				...
				...
				...
...	...	...	...	...
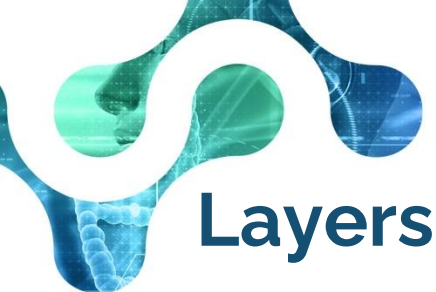
Layers

Conv2D

- **Input a 3D Tensor (channels_in, height_in, width_in)**
- **Output a 3D Tensor (channels_out, height_in, width_out)**

- **channels_out = num_filters**
- **with padding="same"**
    - **height_out= height_in**
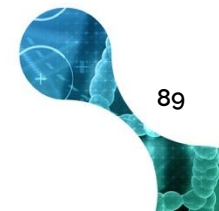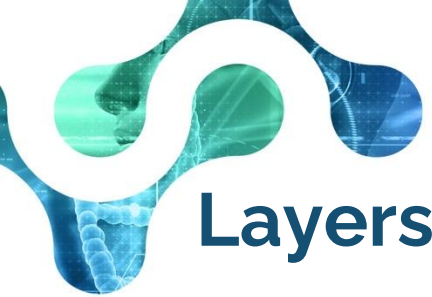   - **width_out= width_in**

Layers

Maxpool2D

- **Input a 3D Tensor (channels_in, height_in, width_in)**

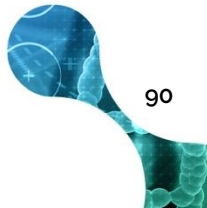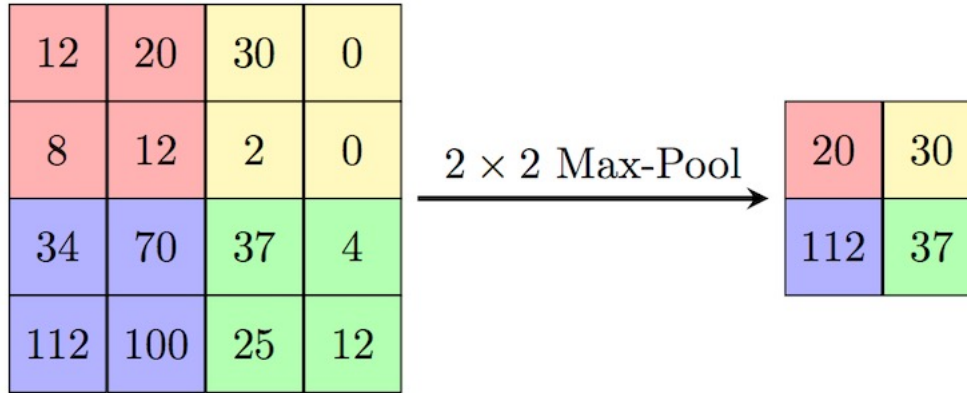- **Output a 3D Tensor (channels_in, height_in/s, width_in/s)**
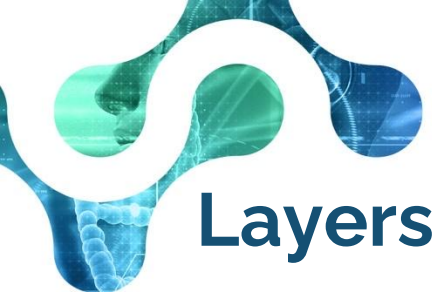
  Normally s=2

- **Parameters: None**

Layers

Maxpool2D

Layers
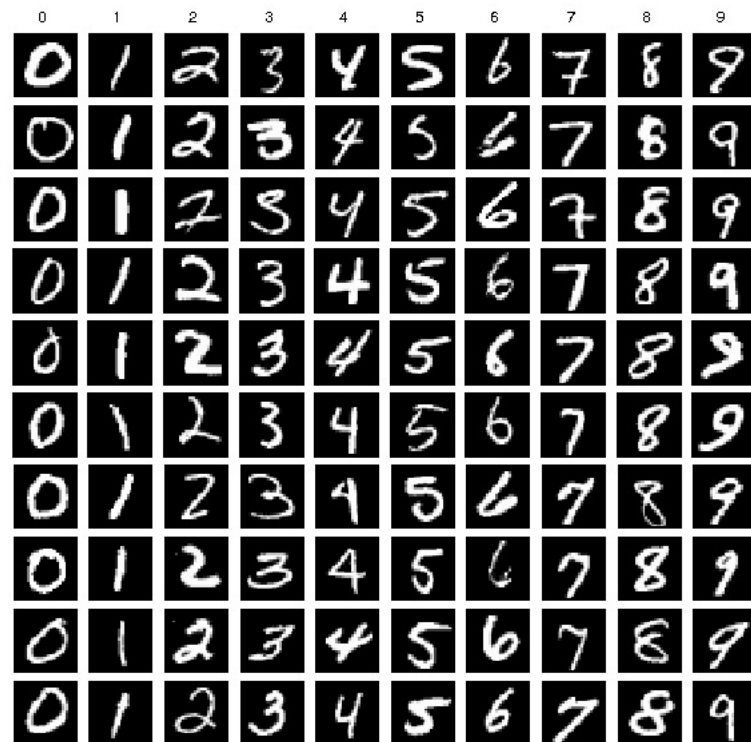
AveragePool2D

2 x 2 Avg-Pool
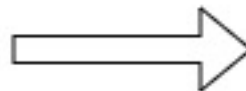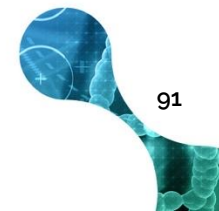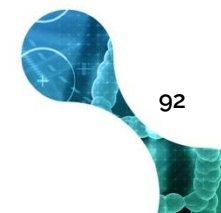
EDDL with Dense topology (MLP)
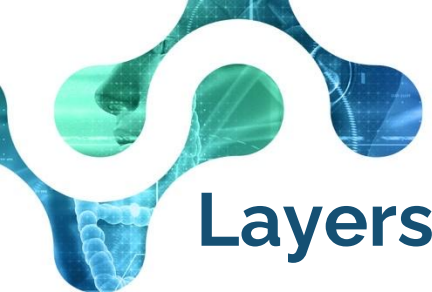

28x28 images (2D)

stored as 784 pixels (1D)

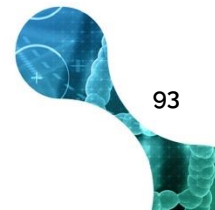10 classes

Layers

```
num_classes = 10

in = eddl.Input([784])                          # 784 components vector
layer = eddl.Reshape(in, [1, 28, 28])           # 28 x 28 gray-level image
layer = eddl.MaxPool(eddl.ReLu(eddl.Conv(layer, 32, [3, 3])))      #(32,14,14)
layer = eddl.MaxPool(eddl.ReLu(eddl.Conv(layer, 64, [3, 3])))      #(64,7,7)
layer = eddl.MaxPool(eddl.ReLu(eddl.Conv(layer, 128, [3, 3])))     #(128,3,3)
layer = eddl.MaxPool(eddl.ReLu(eddl.Conv(layer, 256, [3, 3])))     #(256,1,1)
layer = eddl.Reshape(layer, [-1])               # 256 components vector
out = eddl.Softmax(eddl.Dense(layer, num_classes))
```
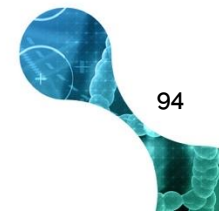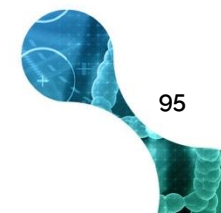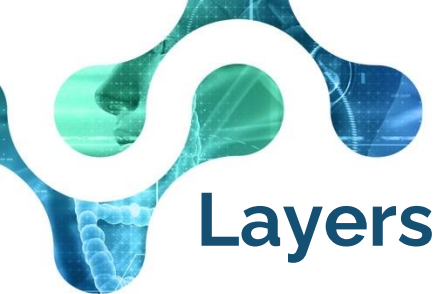
eddl.summary(net)

```
------------------------------------------------------------------------------------
model
------------------------------------------------------------------------------------
input1     | (784)             =>   (784)                0
reshape1   | (784)             =>   (1, 28, 28)          0
conv2d1    | (1, 28, 28)       =>   (32, 28, 28)         320
relu1      | (32, 28, 28)      =>   (32, 28, 28)         0
maxpool2d2| (32, 28, 28)       =>   (32, 14, 14)         0
conv2d2    | (32, 14, 14)      =>   (64, 14, 14)         18496
relu2      | (64, 14, 14)      =>   (64, 14, 14)         0
maxpool2d4| (64, 14, 14)       =>   (64, 7, 7)           0
conv2d3    | (64, 7, 7)        =>   (128, 7, 7)          73856
relu3      | (128, 7, 7)       =>   (128, 7, 7)          0
maxpool2d6| (128, 7, 7)        =>   (128, 3, 3)          0
conv2d4    | (128, 3, 3)       =>   (256, 3, 3)          295168
relu4      | (256, 3, 3)       =>   (256, 3, 3)          0
maxpool2d8| (256, 3, 3)        =>   (256, 1, 1)          0
reshape2   | (256, 1, 1)       =>   (256)                0
dense1     | (256)             =>   (10)                 2570
softmax5   | (10)              =>   (10)                 0
------------------------------------------------------------------------------------
```
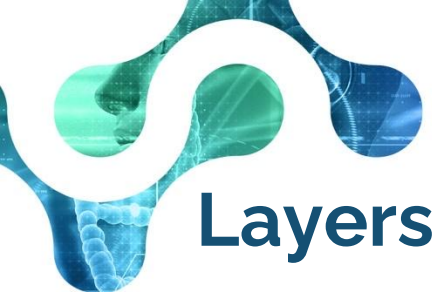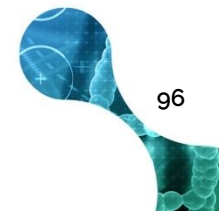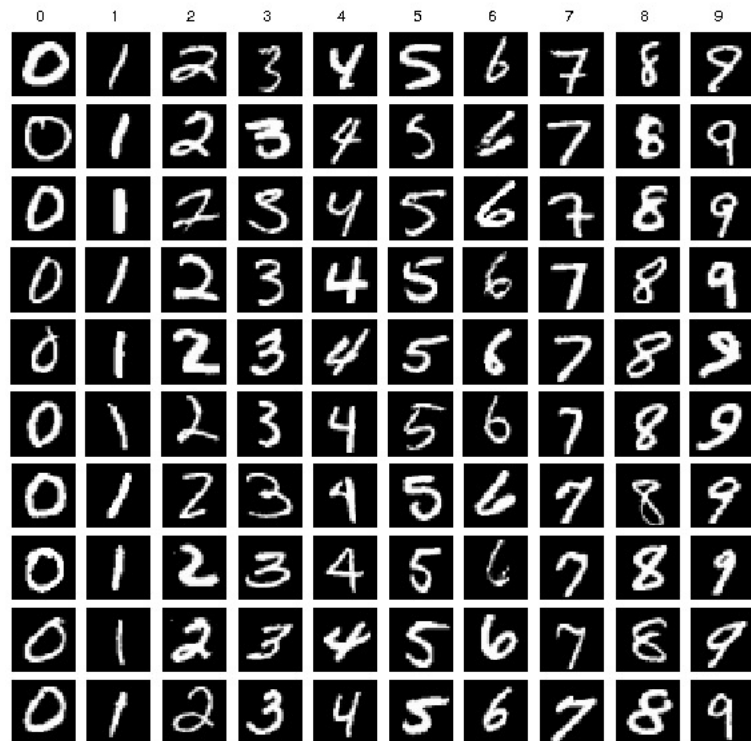
Layers

input1 → reshape1 → conv2d1 → relu1 → maxpool2d2 → conv2d2 → relu2 → maxpool2d4 → conv2d3 → relu3 → maxpool2d6 → conv2d4 → relu4 → maxpool2d8 → reshape2 → dense1 → softmax5

Layers

Recurrent Layers:

- Vanilla RNN
- LSTM
- GRU

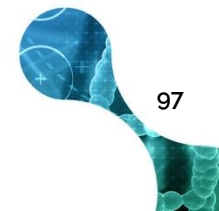All accept a 1D tensor (vector) as Input and returns 1D vector as Output

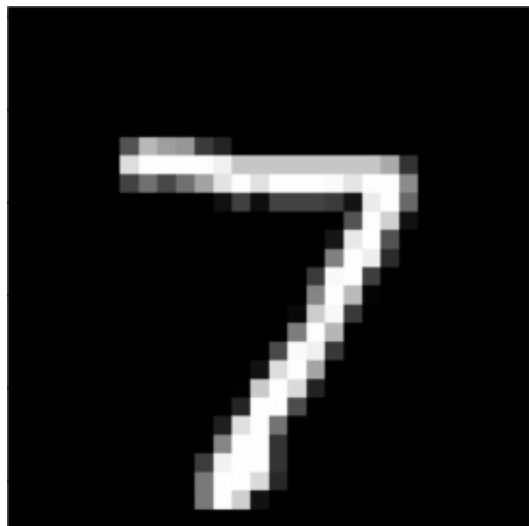# EDDL with Dense topology (MLP)

28x28 images (2D)

stored as 784 pixels (1D)

10 classes

EDDL with Dense topology (MLP)

s
e
q
u
e
n
c
e

We are going to consider the digits as a sequence of 28 rows of 28 pixels

Then the recurrent layers expect to receive a 1D tensor of 28 components

Layers

num_classes = 10

```
in = eddl.Input([28])
layer = eddl.ReLu(eddl.Dense(in, 32))
layer = eddl.LSTM(layer, 128)
out = eddl.Softmax(eddl.Dense(layer, num_classes))
```
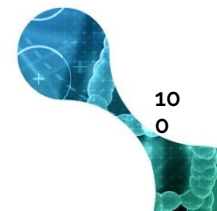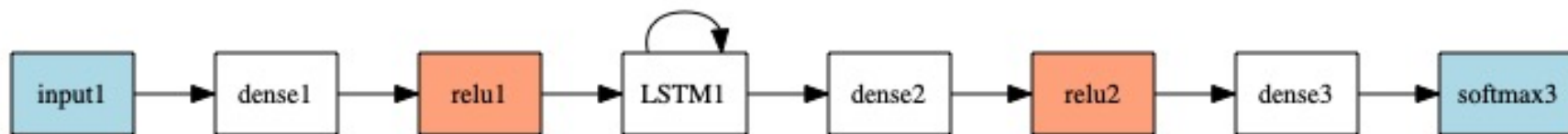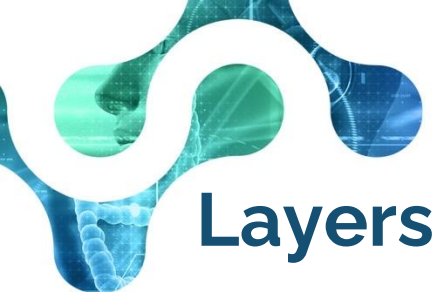
…

```
x_train.reshape_([x_train.shape[0], 28, 28]).     # (N, sequence length, dim)
```

Layers

Thank you!

Roberto Paredes

rparedes@prhlt.upv.es