EVERYTHING YOU NEED TO KNOW TO BUILD YOUR FIRST CONVOLUTIONAL NEURAL NETWORK (CNN)

TARGETED PIECES OF KNOWLEDGE

- Linear regression
- Activation function
- Multi-Layers Perceptron (MLP)
- Stochastic Gradient Descent (SGD)
- Back-propagation
- Convolution
- Pooling (or Sub-sampling)
- Convolutional Neural Networks (CNN)
- Features maps

- Dropout
- Batch Normalization

NOTATION

$\{x, y\}$: a training example (x the input, y the label)

x: a scalar

$\boldsymbol{x}$: a vector

$\boldsymbol{X}$: a matrix

$\boldsymbol{W}, \theta$: network weights

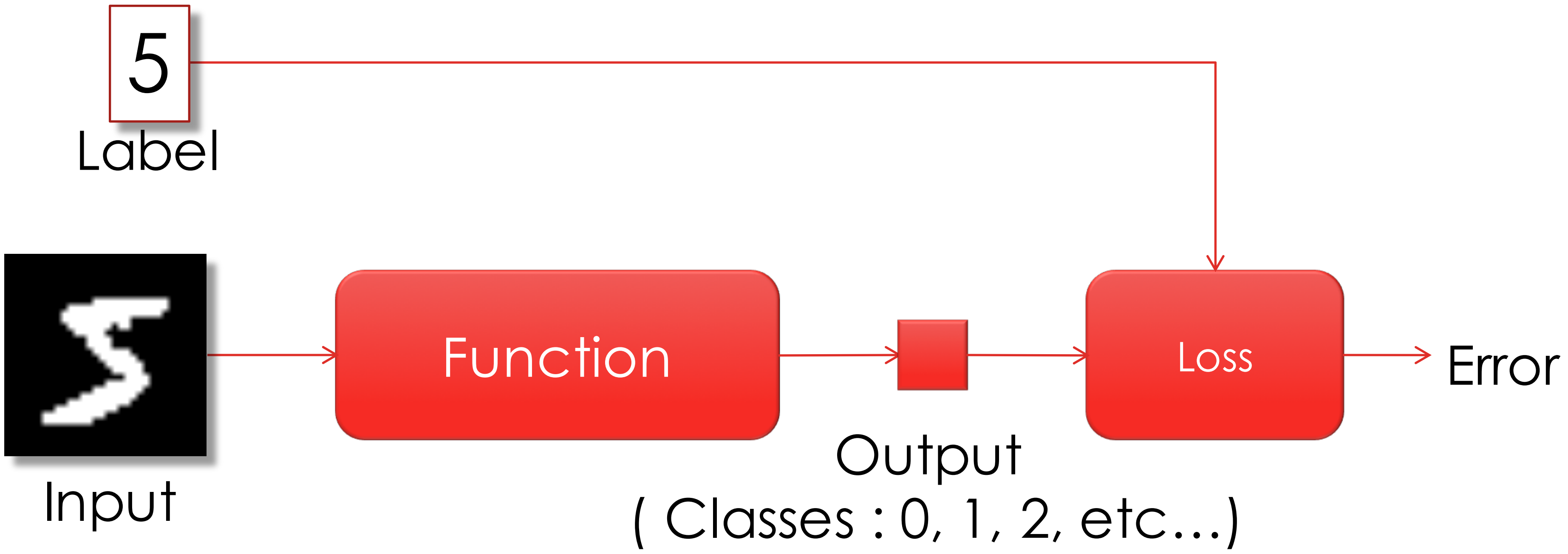
$J(\theta)$: a loss function

MNIST DATASET

Dataset of handwritten digits.
60.000 training data and 10.000 test data.
Digits are size-normalized and centered in fixed-size images.

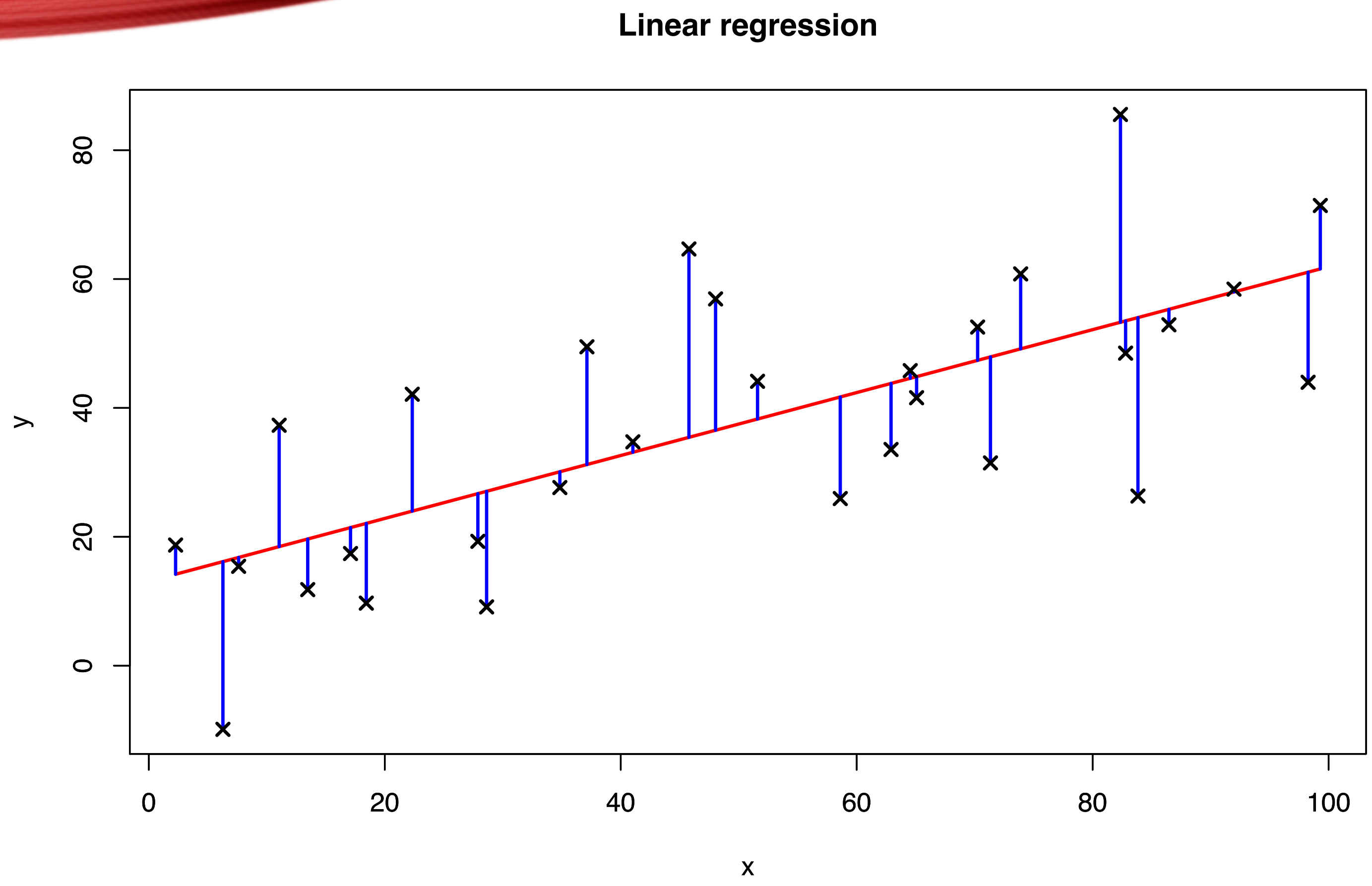
Easy dataset for beginners in machine learning.

SUPERVISED LEARNING



OUR FIRST NEURAL NETWORK

LINEAR REGRESSION



Linear function:

$$f(x, \boldsymbol{w}) = w_0 + w_1 x$$

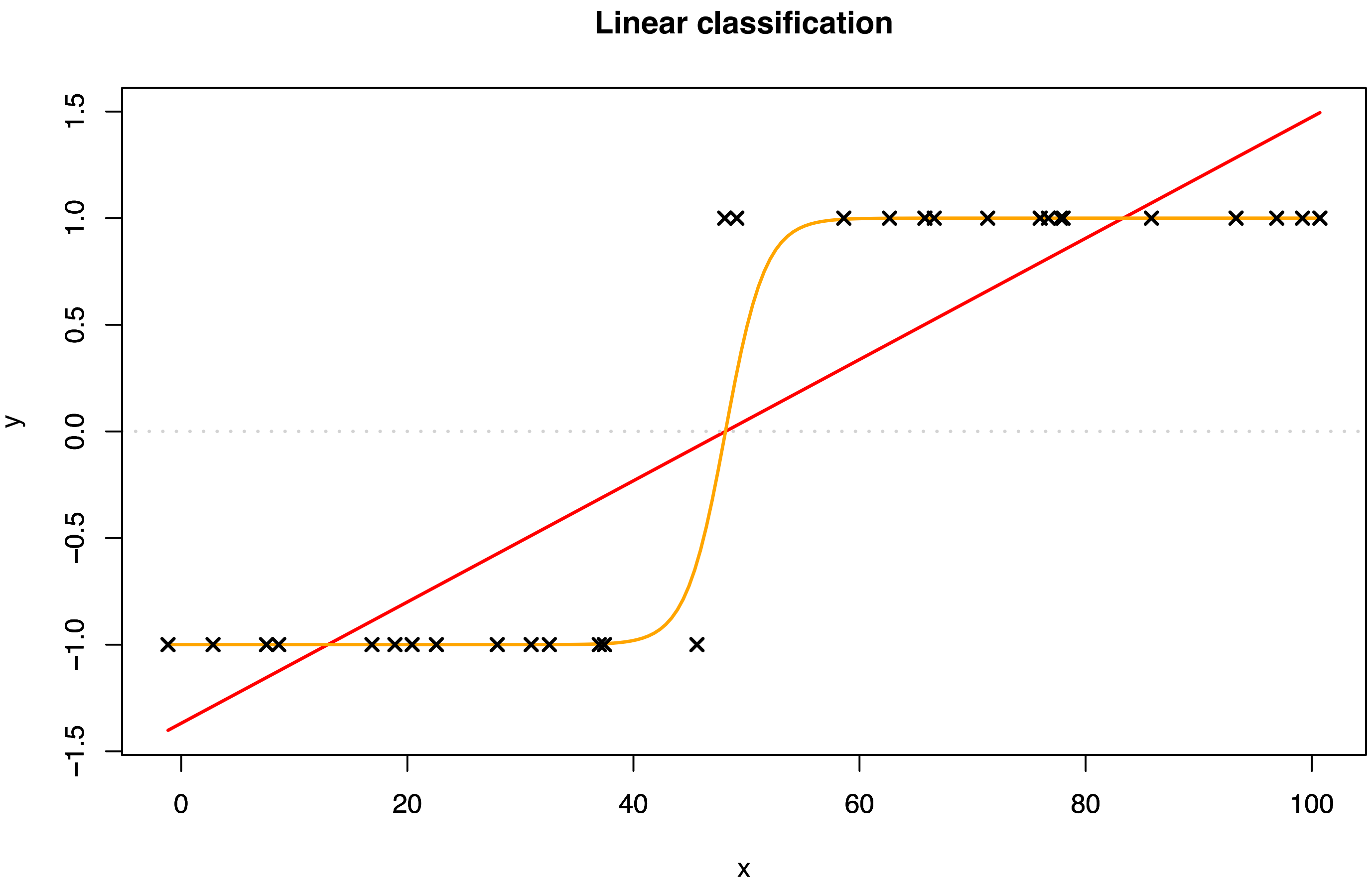
Objective: find $w_0, w_1 = \boldsymbol{w}$ which minimize the error

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i, \boldsymbol{w}) - y_i)^2$$

Animation of the optimization problem

https://jalammar.github.io/visual-interactive-guide-basics-neural-networks/#train-your-dragon

CLASSIFICATION FUNCTION



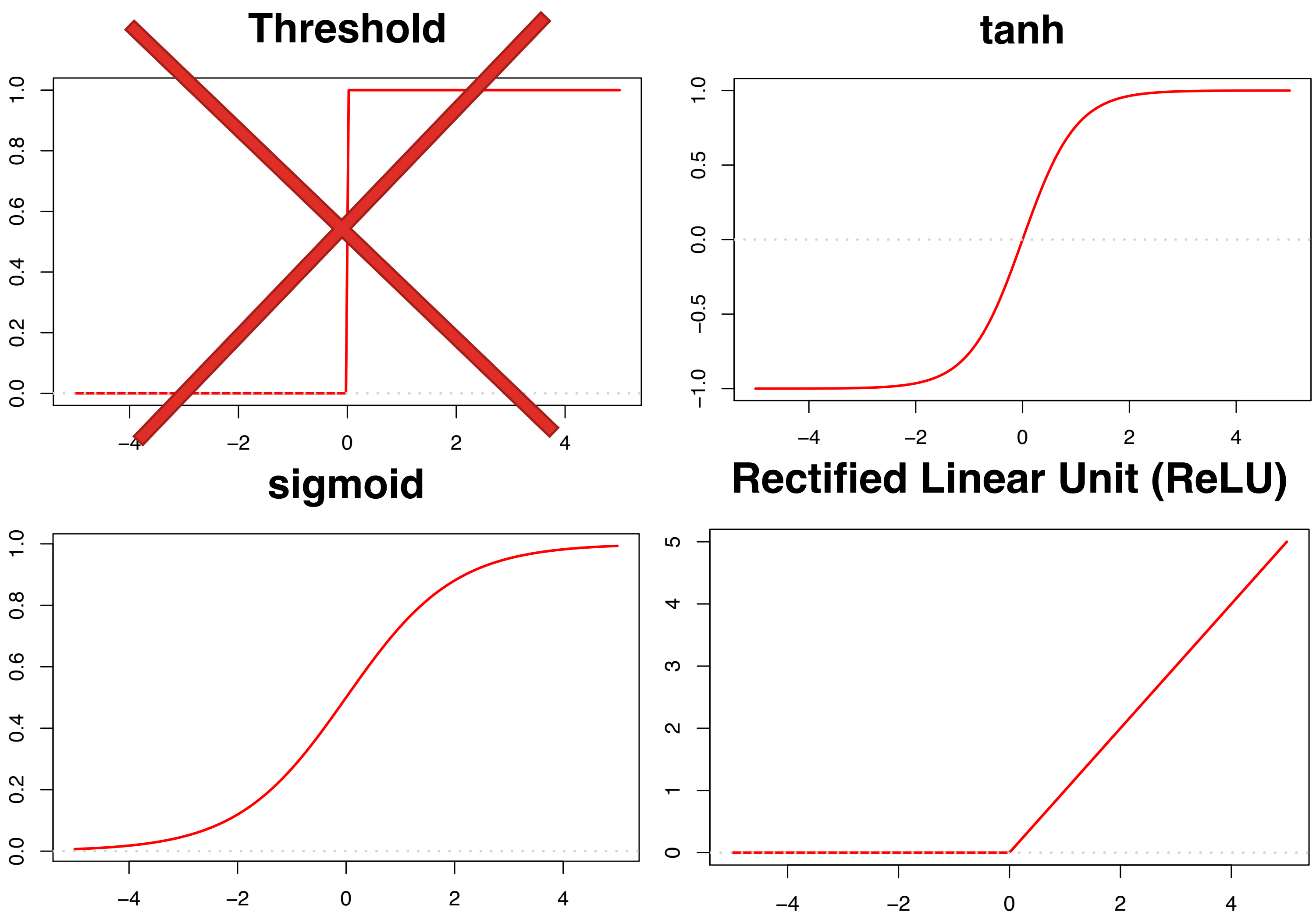
Binary classification:

$$f(x, \boldsymbol{w}) \in \{-1; +1\}$$

Using a non linearity function

$$f(x, \boldsymbol{w}) = \begin{cases} 1 \; if \; \tanh(w_0 + w_1 x) \geq 0 \\ -1 \; otherwise \end{cases}$$

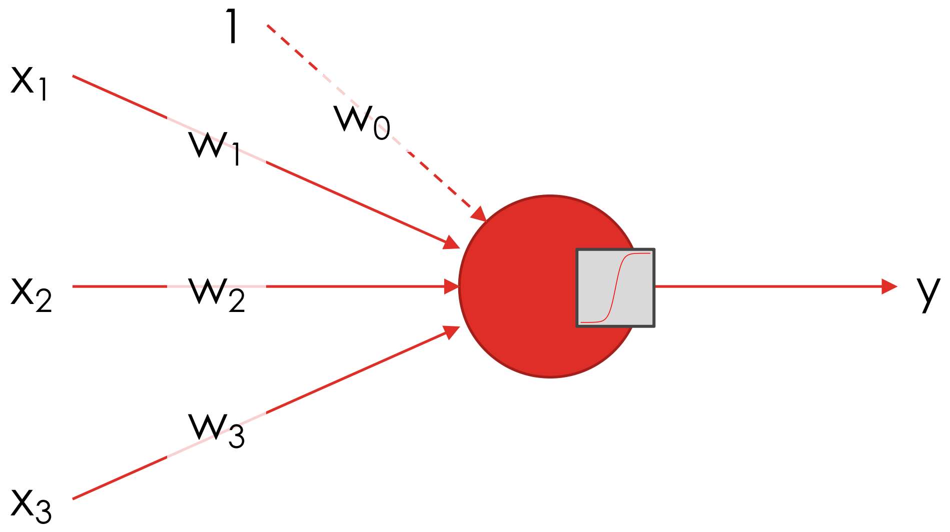
ACTIVATION FUNCTION



- Threshold
- Tanh
- Sigmoïd
- Recitified Linear Unit (ReLU)

- Leaky ReLU
- PReLU
- Etc…

PERCEPTRON



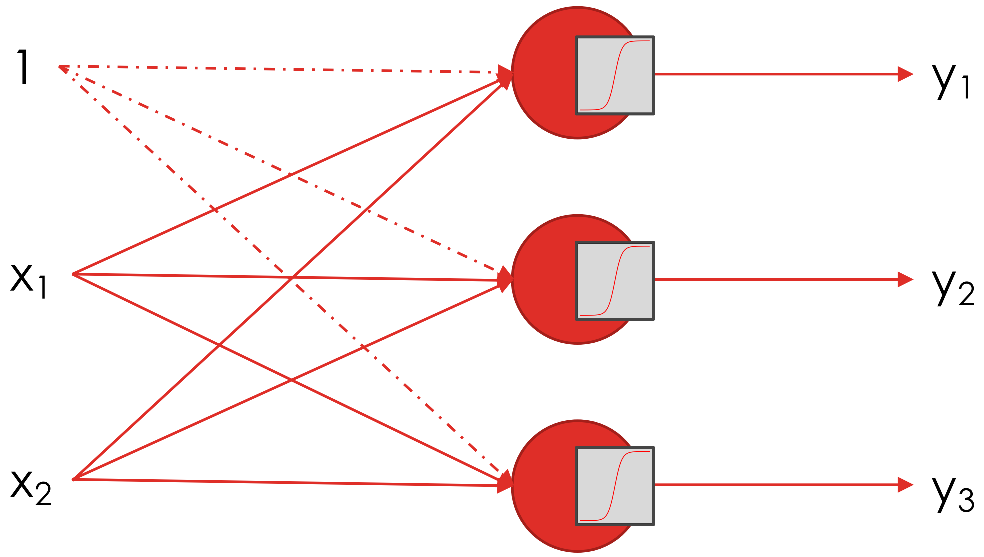
If $h(x)$ is an activation function, then a perceptron if define as follows:

$$F(\boldsymbol{x}, \boldsymbol{w}) = h(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3)$$

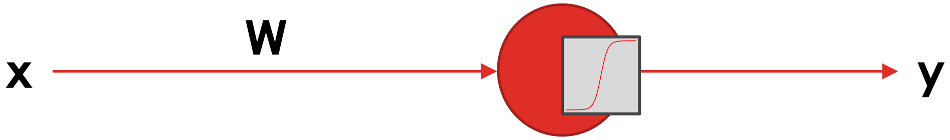
$$= h\left(\sum w_i x_i\right)$$

$$= h(\boldsymbol{w} \cdot \boldsymbol{x}^T)$$

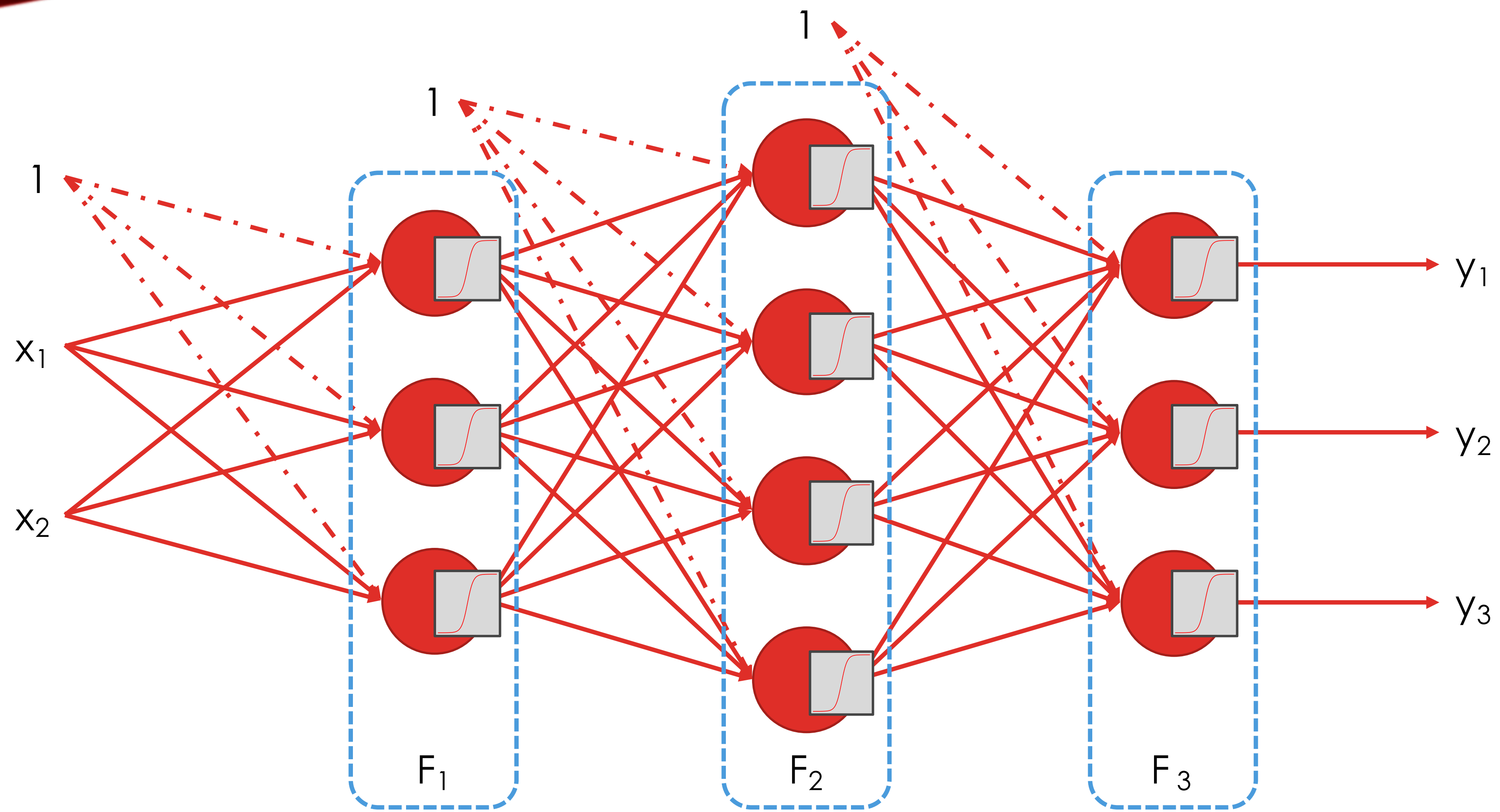
FIRST LAYER OF NEURONES



$$F(\mathbf{x},\mathbf{W}) = h(\mathbf{x}^t \times \mathbf{W}) = h(\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}^t \times \begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}) = h(\begin{bmatrix} w_{01} + x_1 w_{11} + x_2 w_{21} \\ w_{02} + x_1 w_{12} + x_2 w_{22} \\ w_{03} + x_1 w_{13} + x_2 w_{23} \end{bmatrix}^t) = h(\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}^t) = \begin{bmatrix} h(y_1) \\ h(y_2) \\ h(y_3) \end{bmatrix}^t$$



MLP: MULTI LAYER PERCEPTRON

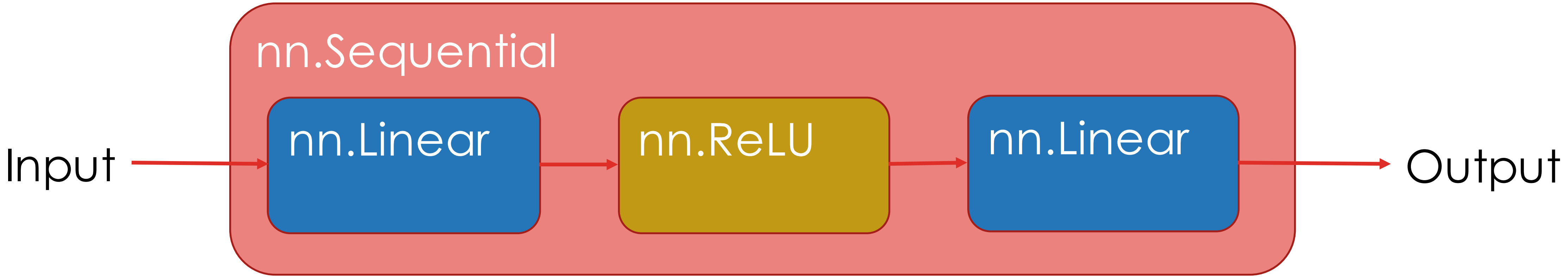


$$F_3(F_2(F_1(\boldsymbol{x}, \boldsymbol{W}_1), \boldsymbol{W}_2), \boldsymbol{W}_3) = F_3\left(F_2\left(F_1(\boldsymbol{x})\right)\right) = (F_3 \circ F_2 \circ F_1)(\boldsymbol{x})$$

BUILDING OUR MLP

Torch7 works with modules.

Module is an abstract class which defines fundamental methods necessary for a training a neural network. Modules are serializable.



LOSS FUNCTION FOR CLASSIFICATION

Converting the network outputs into probabilities:

$$f(y = j|\boldsymbol{u}) = \frac{e^{\boldsymbol{u}_j}}{\sum_{j'=1}^{|\boldsymbol{u}|} e^{\boldsymbol{u}_{j'}}}$$

Negative log likelihood:

$$J(\boldsymbol{p}, t) = -\log(p_t)$$

Combination of both:

$$J(\boldsymbol{u}, t) = -\boldsymbol{u}_t + \log(\sum_{j'=1}^{|\boldsymbol{u}|} e^{\boldsymbol{u}_{j'}})$$

Network output:

$\boldsymbol{u} =$	14	11	16

Class probabilities:

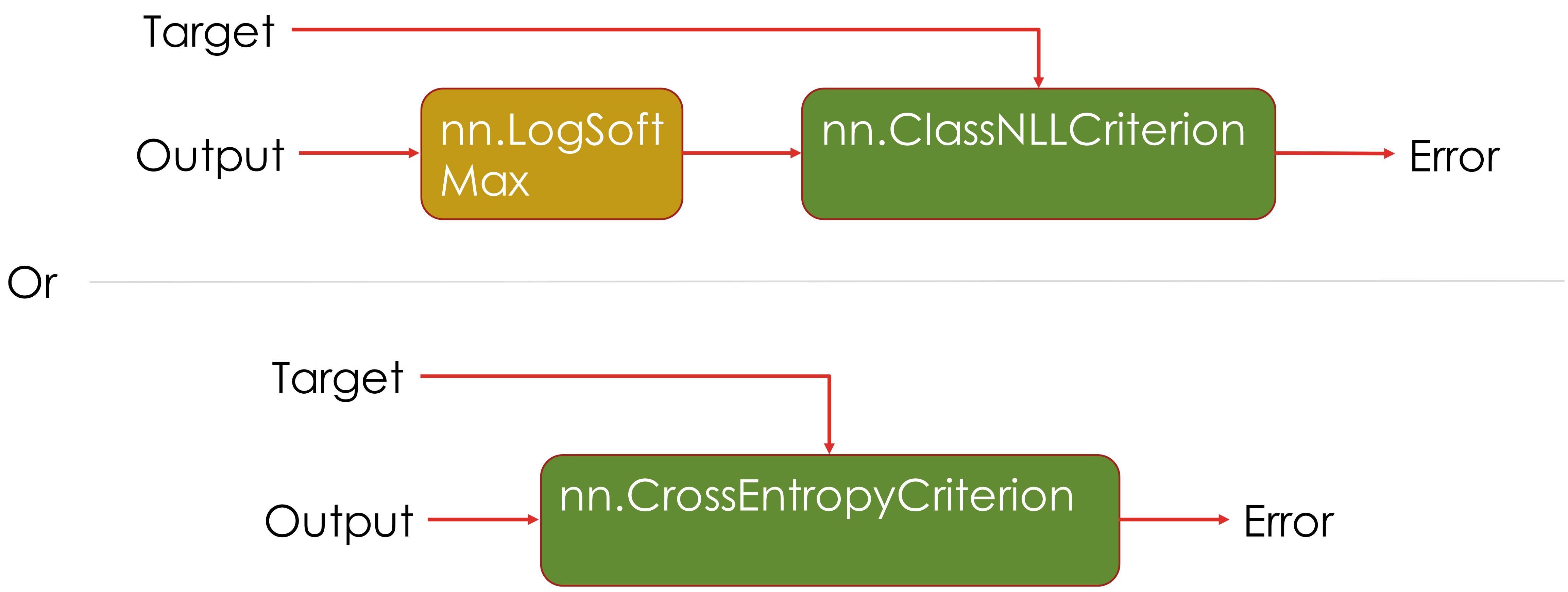
$f(\boldsymbol{u}) =$	.1185	.0059	.8756

Error:

$$J(f(\boldsymbol{u}), 3) = -\log(0.8756)$$
$$= -0.1328$$

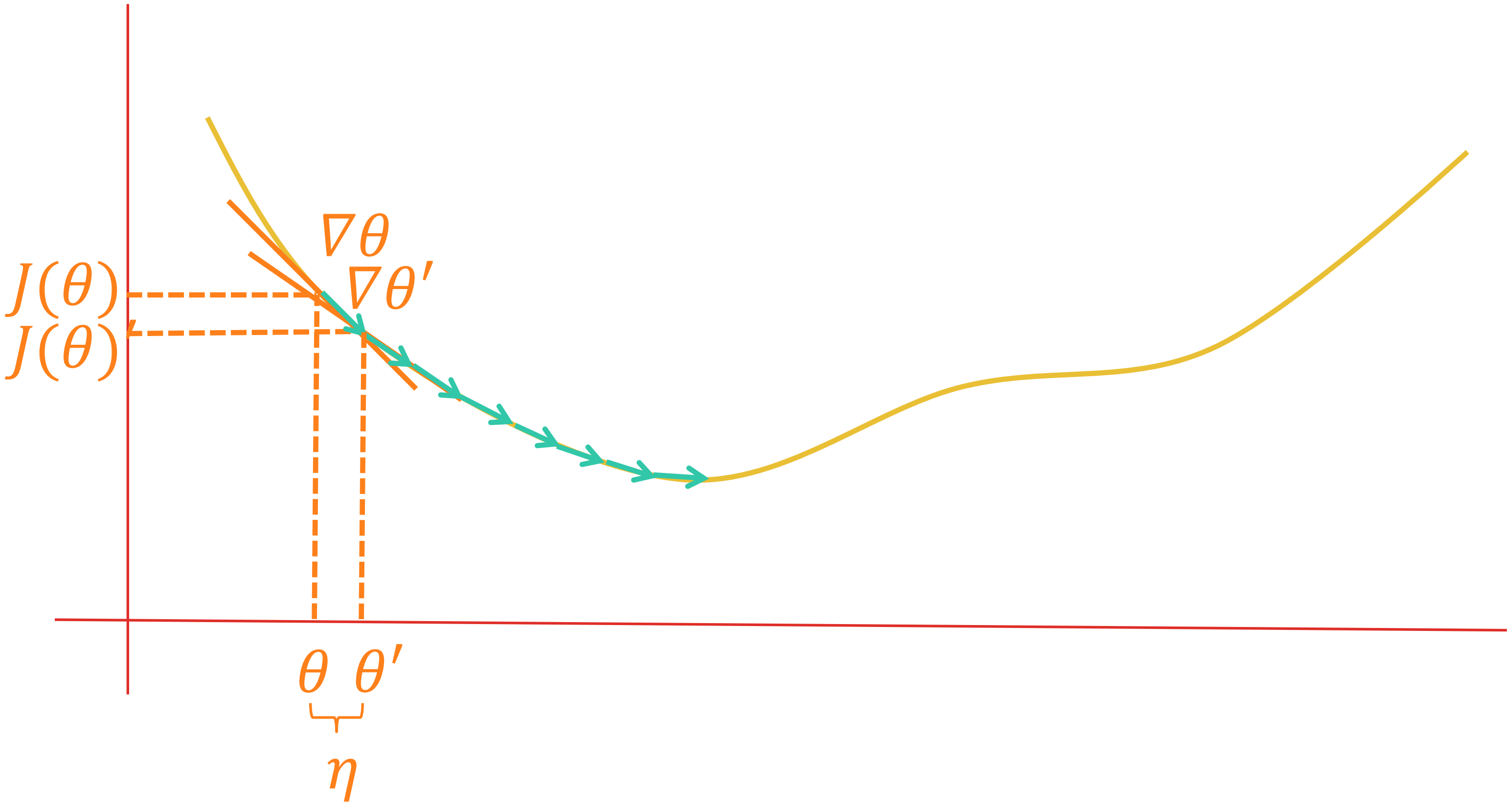
LOSS FUNCTION IN TORCH7

Criterion is a special kind of Module who take to parameters has input



HOW TO TRAIN A NEURAL NETWORK?

GRADIENT DESCENT



Objective: minimizing an objective (loss) function $J(\theta)$

Gradient gives the slope of the function

Updating the parameters θ in the opposite direction of the gradient according to a learning rate η

Repeat until convergence

CHAIN RULE

Composition function:

$$F(x) = (\mathrm{f} \circ g)(x) = f(g(x))$$

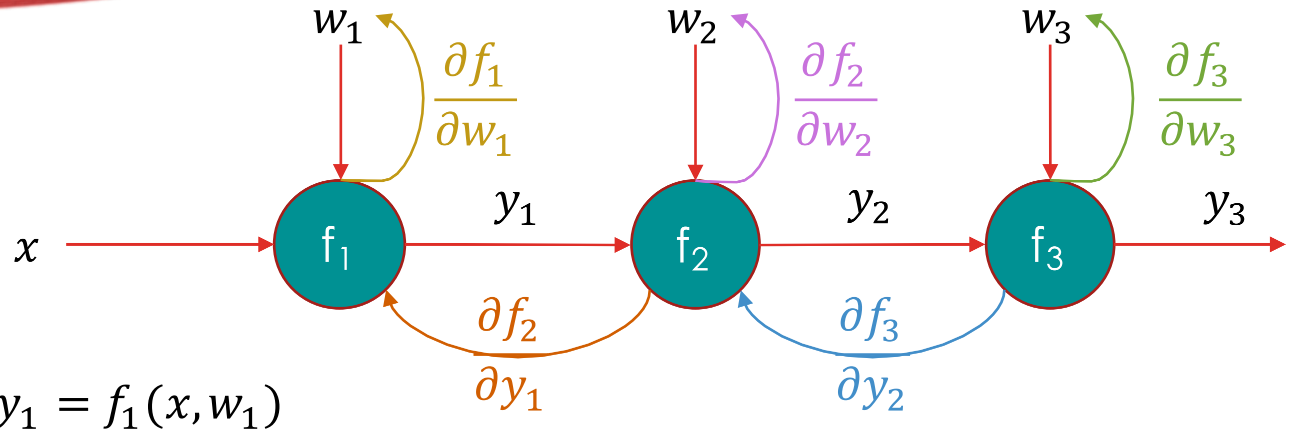
Derivative of a composition function:

$$F'(x) = (f' \circ g)(x) \times g'(x) = f'\big(g(x)\big) \times g'(x)$$

Using the Leibniz's notation:

$$F'(x) = \frac{\partial F(x)}{\partial x} = \frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \times \frac{\partial g(x)}{\partial x}$$

BACK-PROPAGATION



$$y_1 = f_1(x, w_1)$$

$$y_2 = f_2(y_1, w_2) = f_2(f_1(x, w_1), w_2)$$

$$y_3 = f_3(y_2, w_3) = f_3(f_2(y_1, w_2), w_3) = f_3(f_2(f_1(x, w_1), w_2), w_3)$$

$$\nabla_{w_3} y_3 = \frac{\partial y_3}{\partial w_3} = \frac{\partial f_3(y_2, w_3)}{\delta w_3}$$

$$\nabla_{w_2} y_3 = \frac{\partial y_3}{\partial w_2} = \frac{\partial f_3(y_2, w_3)}{\partial w_2} = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial y_2}{\partial w_2} = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_2}$$

$$\nabla_{w_1} y_3 = \frac{\partial y_3}{\delta w_1} = \frac{\partial f_3(y_2, w_3)}{\partial w_1} = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial y_2}{\partial w_1} = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_1} = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial y_1} \times \frac{\partial y_1}{\partial w_1}$$

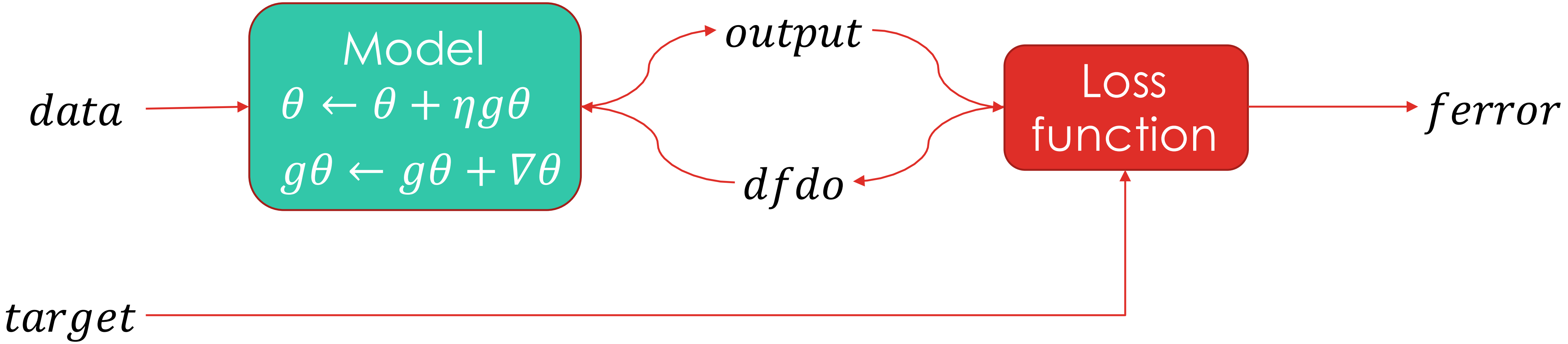
<u>Objective</u>: $\nabla_{w_1,w_2,w_3} y_3 = \nabla_{w_1} y_3; \nabla_{w_2} y_3; \nabla_{w_3} y_3$

$$\nabla_{w_3} y_3 = \frac{\partial f_3(y_2, w_3)}{\partial w_3}$$

$$\nabla_{w_2} y_3 = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_2}$$

$$\nabla_{w_1} y_3 = \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial y_1} \times \frac{\partial f_1(x, w_1)}{\partial w_1}$$

ONE STEP IN TORCH7




```lua
-- Reset gradients
model:zeroGradParameters()

-- Forward
local output = model:forward(data)
local f_error = loss_function:forward(output, target)

-- Backward
local df_do = loss_function:backward(output, target)
model:backward(data, df_do)

-- Update parameters
model:updateParameters(0.01)
```

BATCH GRADIENT DESCENT

Computes the gradient of the cost function for the entire dataset:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta)$$

```
-- Reset gradients
model:zeroGradParameters()

for i=1, trainData:size() do
        -- Forward
        local output = model:forward(trainData.data[i])
        local f_error = loss_function:forward(output, trainData.labels[i])

        -- Backward
        local df_do = loss_function:backward(output, trainData.labels[i])
        model:backward(trainData.data[i], df_do)
end

-- Update parameters
model:updateParameters(0.01)
```

STOCHASTIC GRADIENT DESCENT

Performs a parameter update for *each* training example $x^{(i)}, y^{(i)}$:

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta, x^{(i)}, y^{(i)})$$

```
-- Create a random permutation
shuffle = torch.randperm(trainData:size())

for i=1, trainData:size() do
        -- Reset gradients
        model:zeroGradParameters()

        -- Forward
        local output = model:forward(trainData.data[shuffle[i]])
        local f_error = loss_function:forward(output, trainData.labels[shuffle[i]])

        -- Backward
        local df_do = loss_function:backward(output, trainData.labels[shuffle[i]])
        model:backward(trainData.data[shuffle[i]], df_do)

        -- Update parameters
        model:updateParameters(0.01)
end
```

MINI-BATCH SGD

Takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta, x^{(i:i+n)}, y^{(i:i+n)})$$

```
for i=1, trainData:size(), batchSize do
        -- Reset gradients
        model:zeroGradParameters()

        -- Create batch
        batch = getBatch(trainData, batchSize)

        -- Forward
        local output = model:forward(batch.inputs)
        local f_error = loss_function:forward(output, batch.targets)

        -- Backward
        local df_do = loss_function:backward(output, batch.targets)
        model:backward(batch.inputs, df_do)

        -- Update parameters
        model:updateParameters(0.01)
end
```

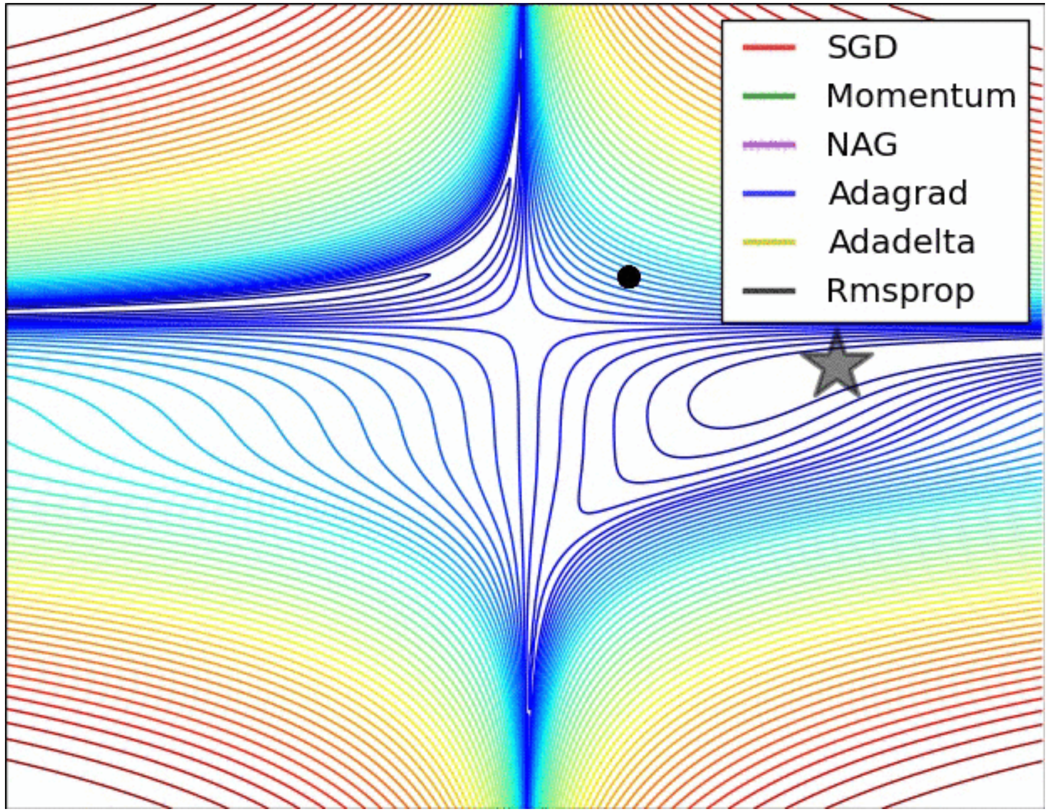
SGD OPTIMIZATION ALGORITHMS

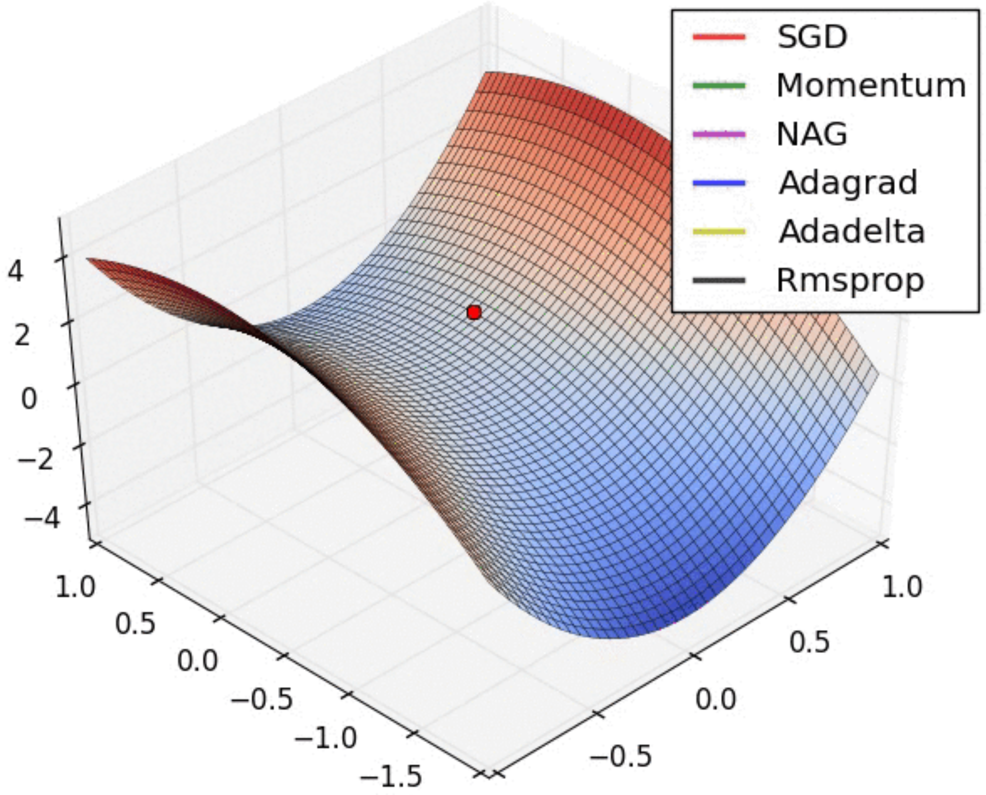
Momentum: adds a fraction of the previously computed gradient (gives inertia to the gradient)

$$v_t \leftarrow \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta \leftarrow \theta - v_t$$

NAG: extension of momentum
Adagrad: adapts the learning rate to each parameters individually
Adadelta: extension of Adagrad
RMSprop: another extension of Adagrad
Adam: takes into account the mean and variance of gradients
Etc…

GRADIENT DESCENT ILLUSTRATION





See: http://sebastianruder.com/optimizing-gradient-descent/index.html#whichoptimizertouse

PACKAGE OPTIM IN TORCH7

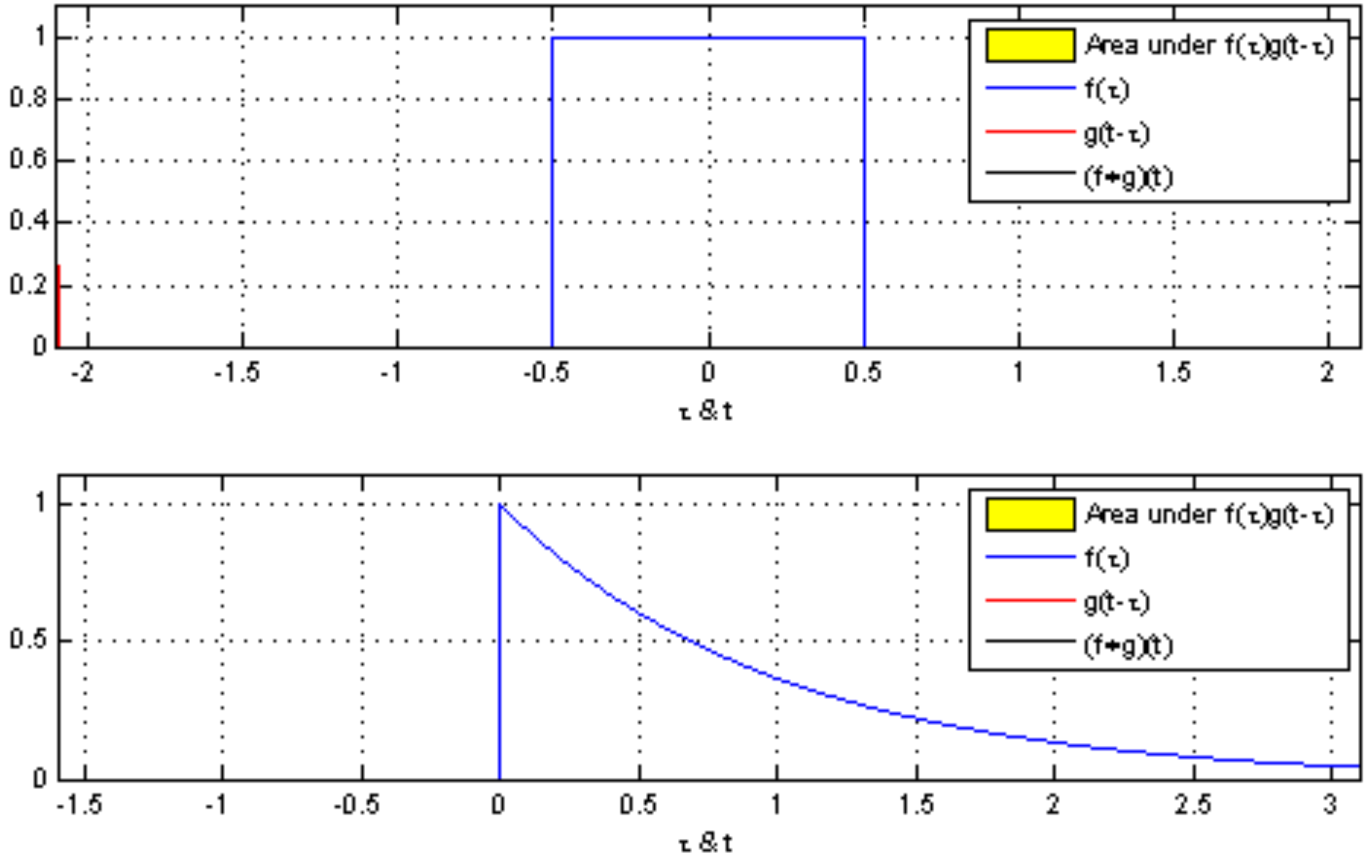
Torch package providing several optimization algorithms.
Easy to use, easy to switch from one optimizer to another.

https://github.com/torch/optim

CONVOLUTIONAL NEURAL NETWORK

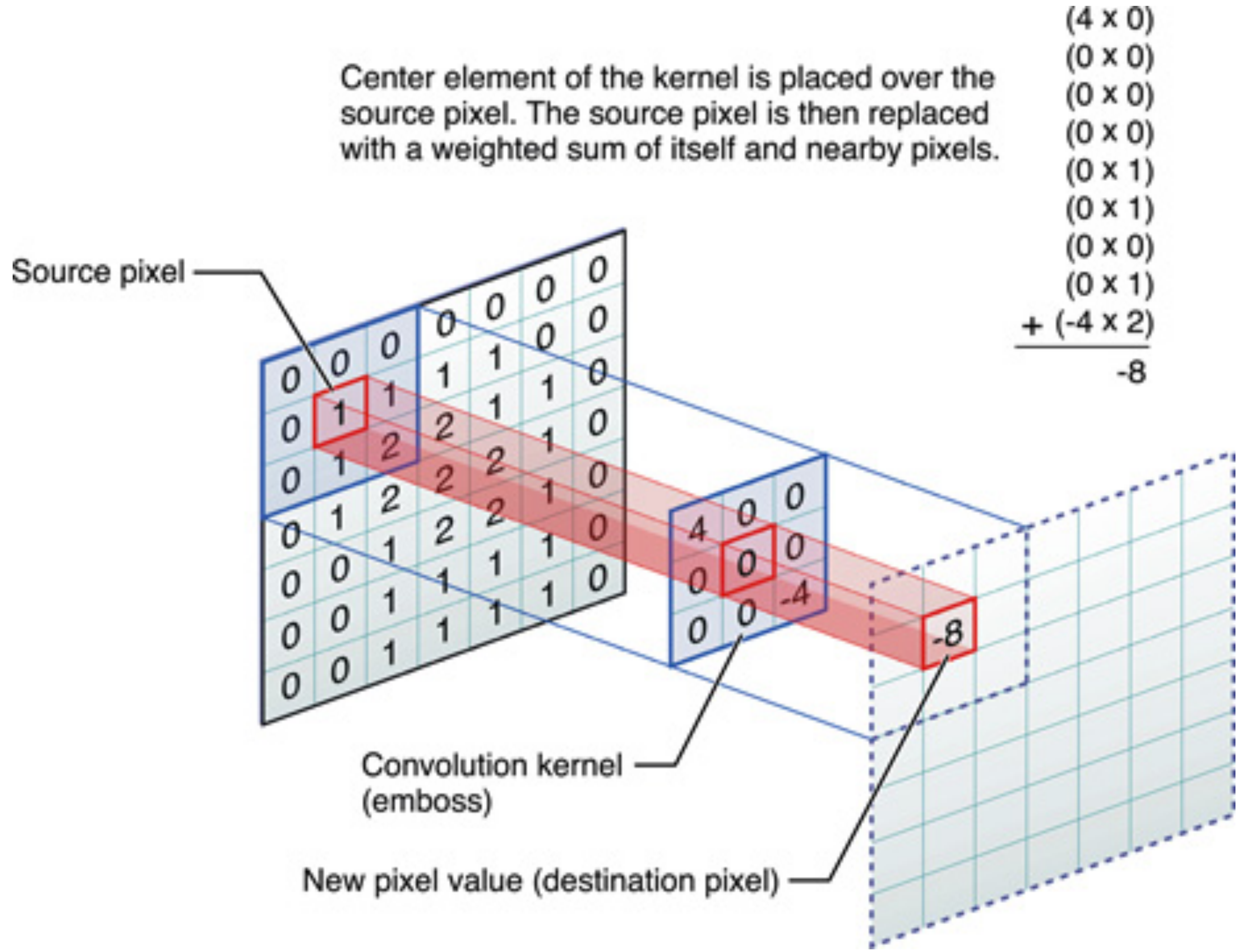
CONVOLUTION

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(t) \cdot g(x - t)dt$$

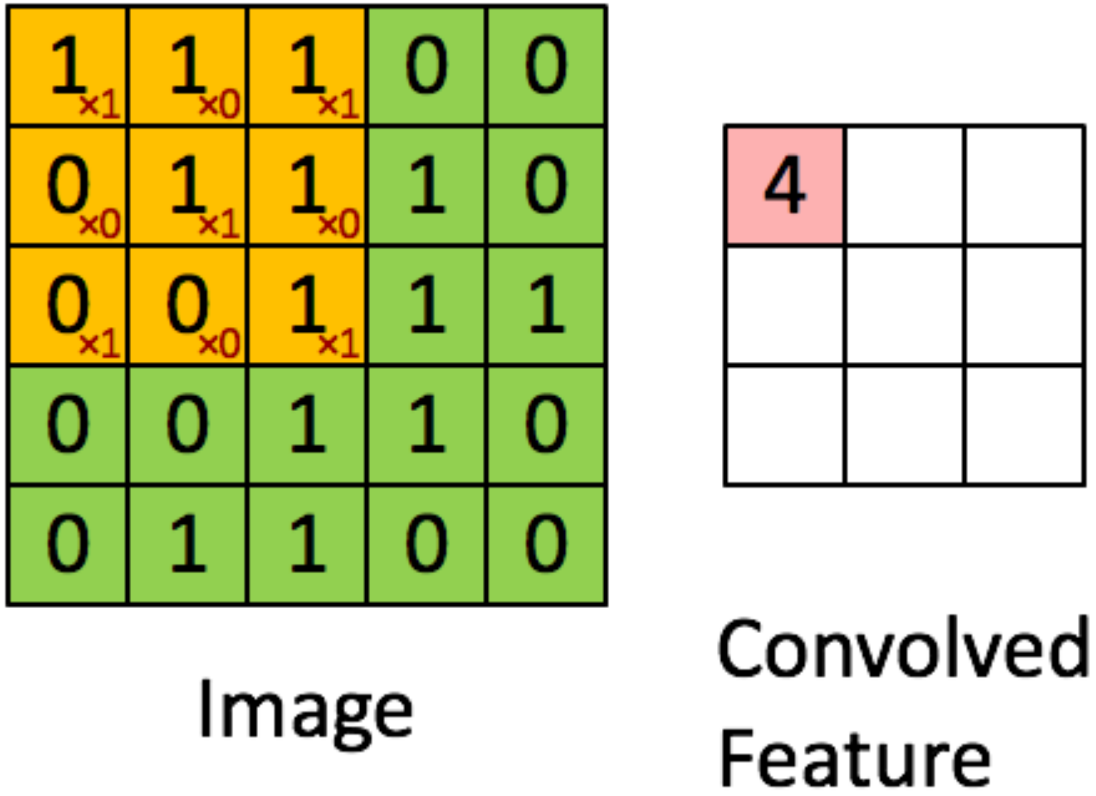


DISCRETE CONVOLUTION

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(n-m) \cdot g(m)$$



SLIDING MASK



Convolution tool from Rémi Emonet: http://dl.heeere.com/convolution/

Convolution layer in Torch7: https://github.com/torch/nn/blob/master/doc/convolution.md

CONVOLUTION EXAMPLE

Original image

0	-1	0
-1	5	-1
0	-1	0

Sharpen

-2	-1	0
-1	1	1
0	1	2

Emboss

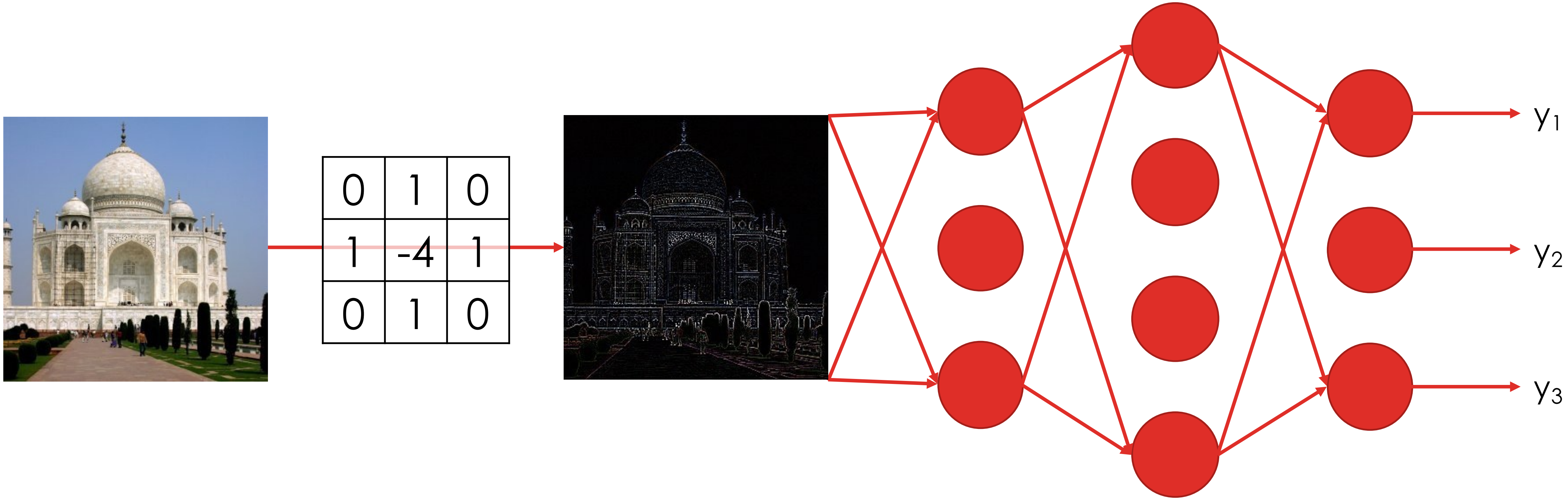
1	1	1
1	1	1
1	1	1

Blur

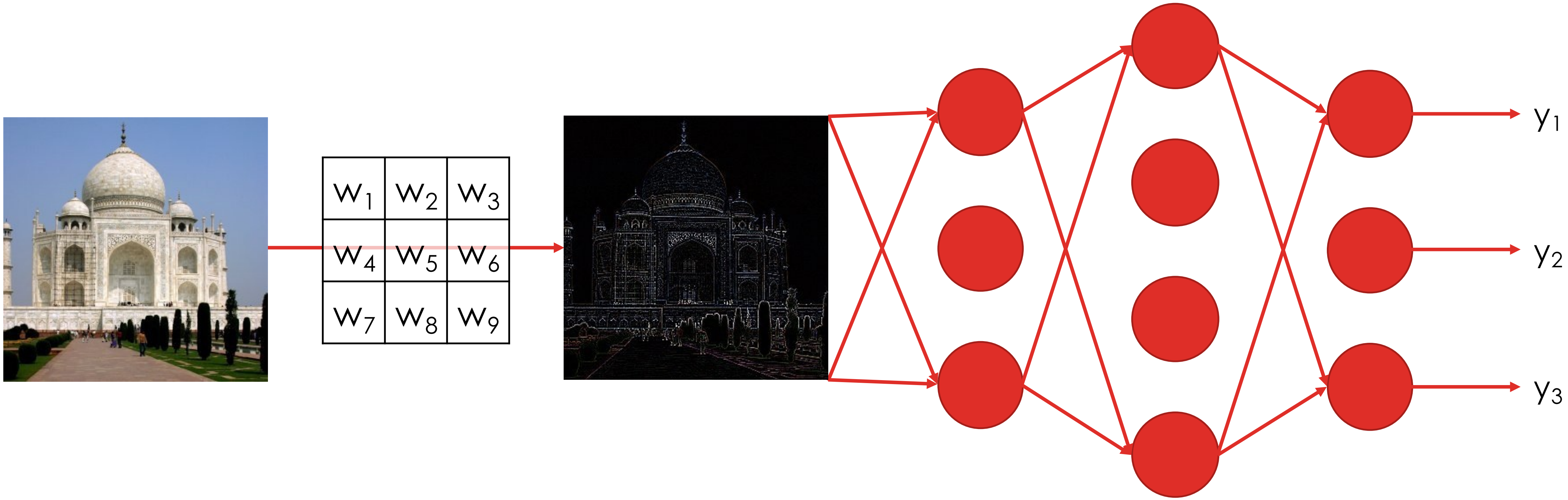
0	1	0
1	-4	1
0	1	0

Edge detect

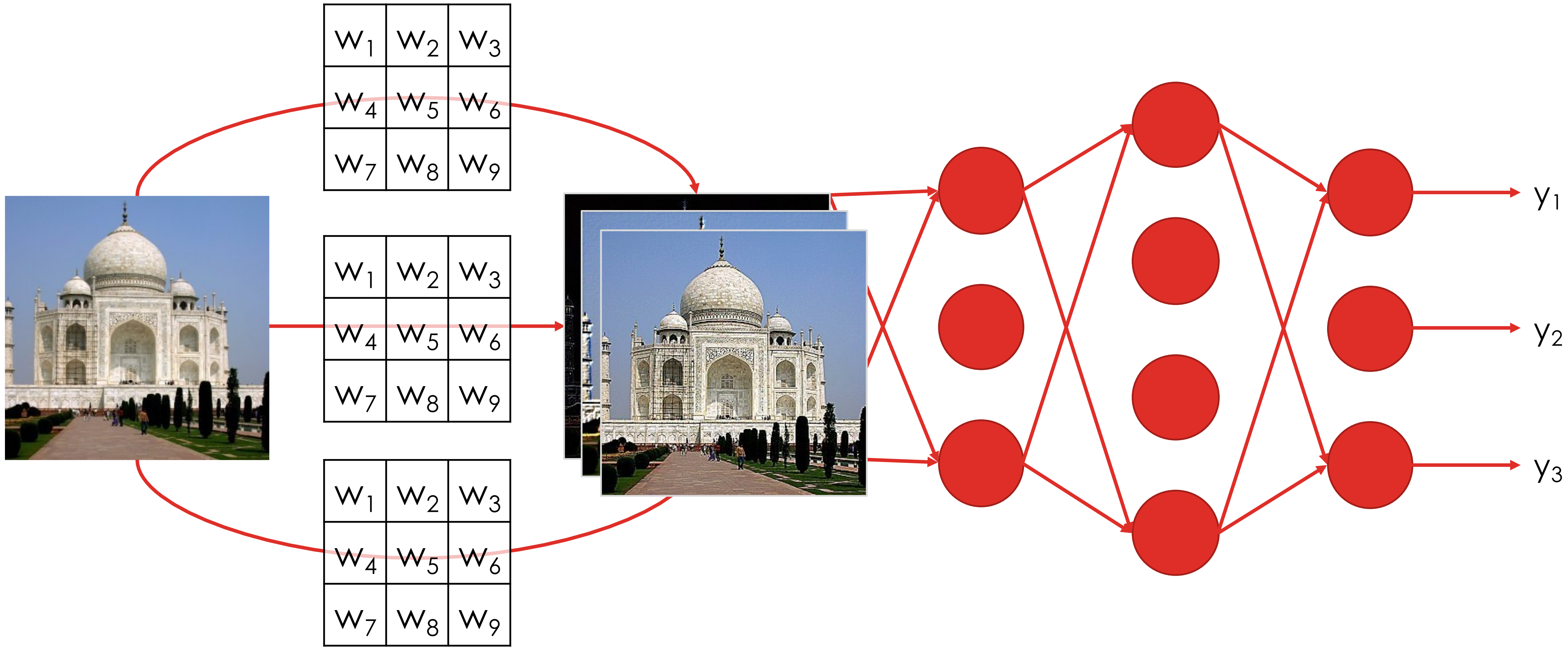
CONVOLUTIONAL NEURAL NETWORK



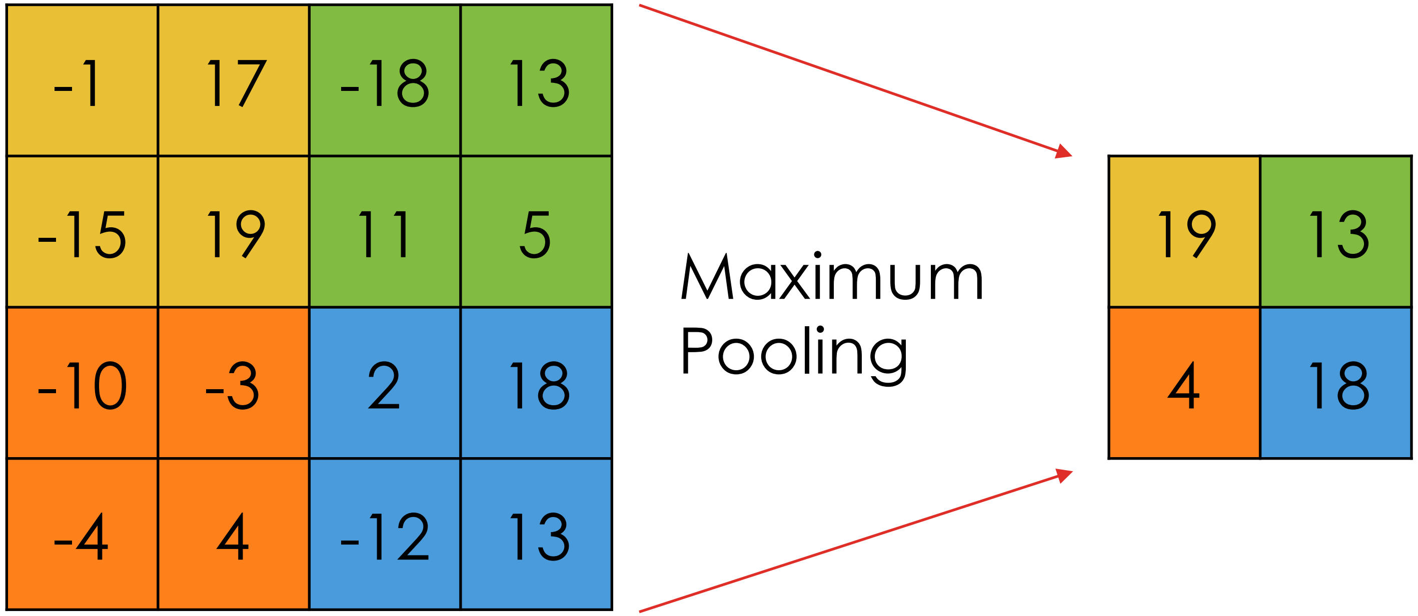
CONVOLUTIONAL NEURAL NETWORK



CONVOLUTIONAL NEURAL NETWORK



POOLING

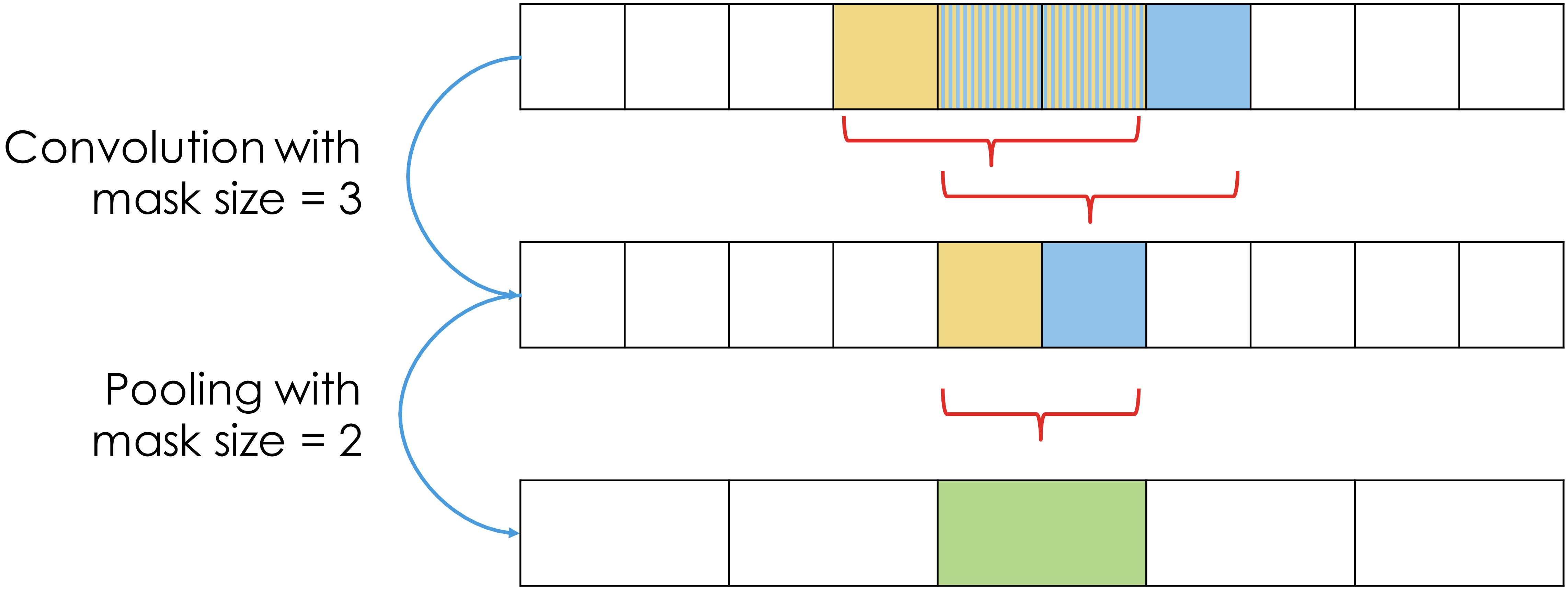


Effect:

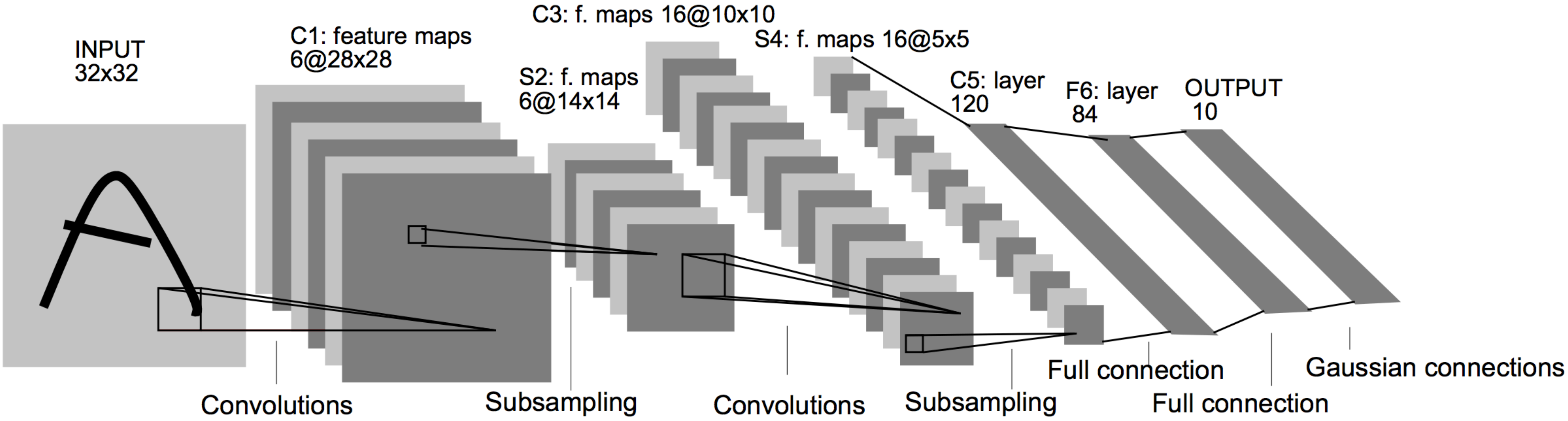
- Reduces the feature map's size
- Increases the field of view

- Average pooling
- Sum pooling
- Stochastic pooling
- Etc …

FIELD OF VIEW



LENET 5



Gradient-based learning applied to document recognition, Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner [1998].

IF WE HAVE TIME…

DROPOUT

During **training**, for each forward pass, randomly set units to 0.

Input

13	9	7	4
4	16	2	5
0	6	8	19
4	7	7	5

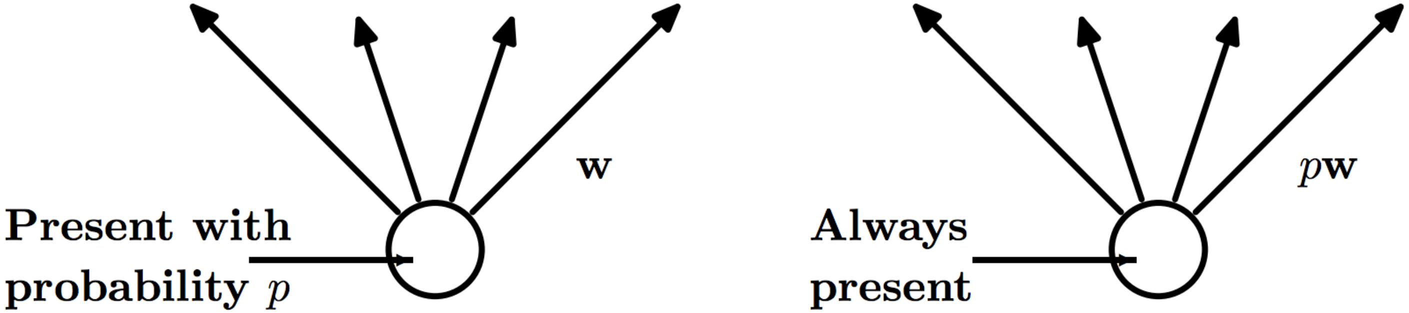
Dropout

Drop factor =0.3

Output

13	0	7	4
4	0	2	0
0	6	8	0
0	0	7	5

At **test** time, keep the same « energy » into the network



(a) At training time

(b) At test time

BATCH NORMALIZATION

During **training**, for each forward pass, normalized the data according to the **mini-batch**

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

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

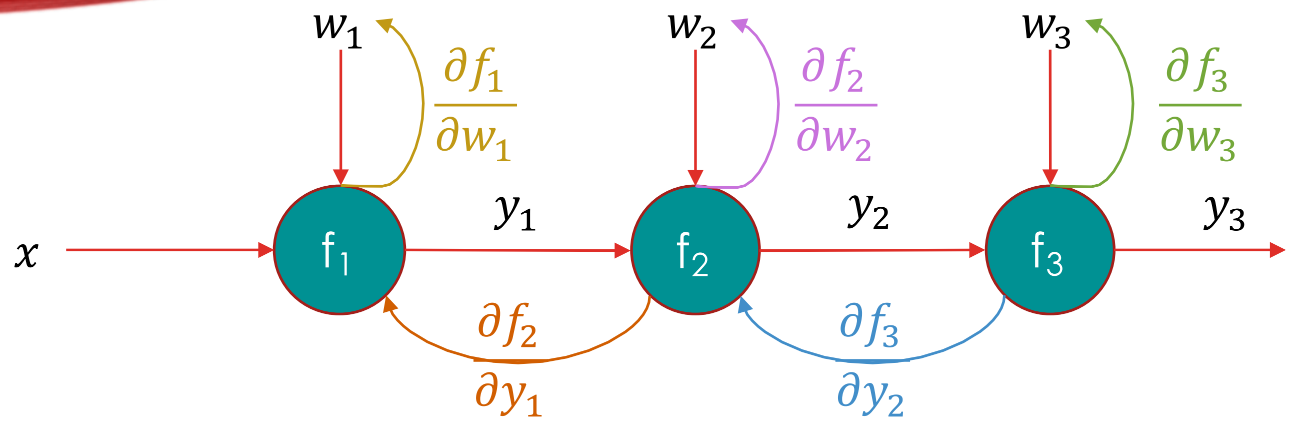
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

CREATING OUR OWN LAYERS



Each module $f(x, w) = y$ have to compute:

- y
- $\partial f / \partial x$
- $\partial f / \partial w$

In Torch7 a new module have to overload 3 functions:

- [output] updateOutput(input)
- [gradInput] updateGradInput(input, gradOutput)
- accGradParameters(input, gradOutput)

Torch7 documentations: http://torch.ch/docs/developer-docs.html

LINKS

TORCH7

Torch7 Documentation:

- Torch7: http://torch.ch/
- Optim package: https://github.com/torch/optim
- Criterions: https://github.com/torch/nn/blob/master/doc/criterion.md
- Convolutional modules: https://github.com/torch/nn/blob/master/doc/convolution.md

Some tutorials code in torch7:

- Torch7 tutorials: https://github.com/torch/tutorials
- Digit classifier: https://github.com/torch/demos/tree/master/train-a-digit-classifier

TUTORIALS

- A Visual and Interactive Guide to the Basics of Neural Networks: https://jalammar.github.io/visual-interactive-guide-basics-neural-networks/
- An overview of gradient descent optimization algorithms: http://sebastianruder.com/optimizing-gradient-descent/index.html
- Artificial Inteligence: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/
- Andrew Ng lesson on coursera: https://www.coursera.org/learn/machine-learning

TARGETED PIECES OF KNOWLEDGE

- Linear regression
- Activation function
- Multi-Layers Perceptron (MLP)
- Stochastic Gradient Descent (SGD)
- Back-propagation
- Convolution
- Pooling (or Sub-sampling)
- Convolutional Neural Networks (CNN)
- Features maps

- Dropout
- Batch Normalization