Software Project

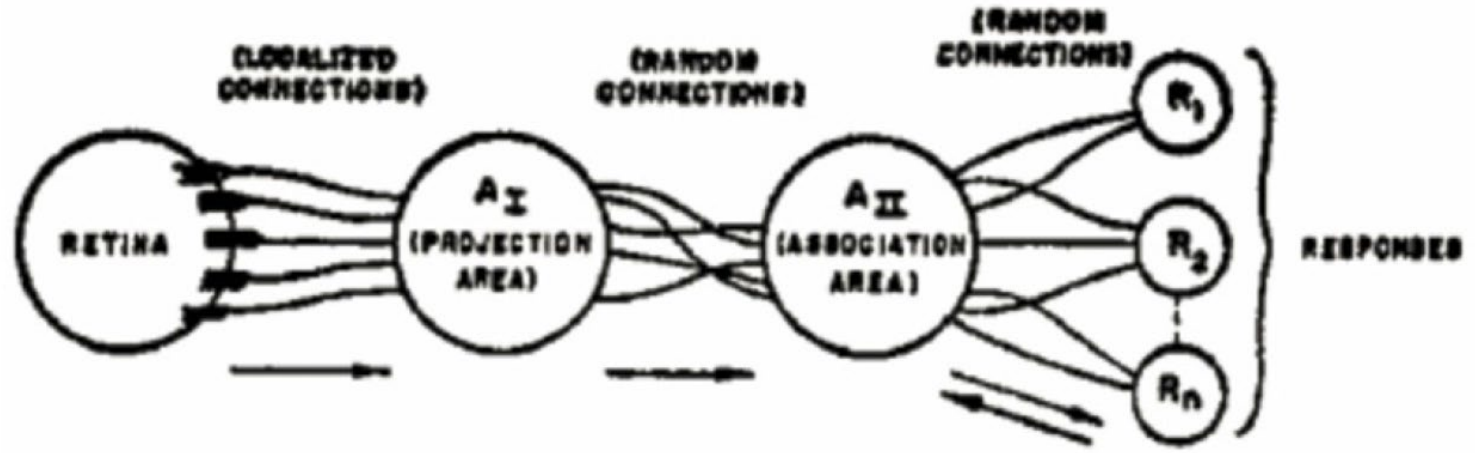
2021-Fall

Prof. Hyunjung Shim

Slide credits: Bhiksha Raj & MIT 6.S191: Introduction to Deep Learning

Perceptron for excitatory or inhibitory

Original perceptron model



- Groups of retina cells (sensors) form A1.
- Groups of A1 cells form A2.
- Signals from A2 cells combine into response cells R.
- All connections can be excitatory or inhibitory.

What Perceptron can do?

Originally it was supposed to represent any Boolean circuit and perform any logic.

- **AND, OR, NOT or AND NOT**

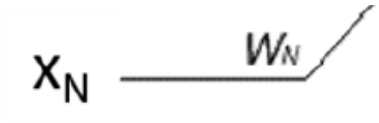
Inputs Weights

$$\mathbf{w} = \mathbf{w} + \eta(d(\mathbf{x}) - y(\mathbf{x}))\mathbf{x}$$

Sequential Learning:

$d(x)$ is the desired output in response to input x

$y(x)$ is the actual output in response to x



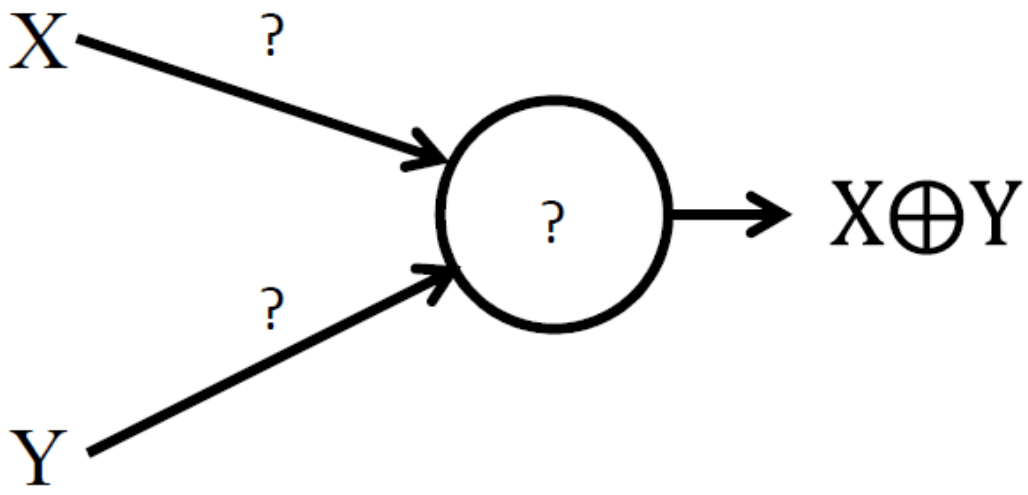
The Perceptron: Boolean task

X

Y

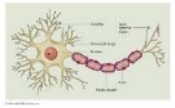
No solution for XOR!

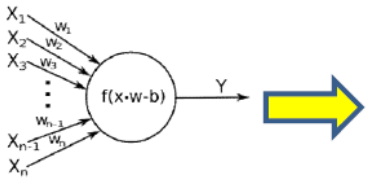
Not universal!

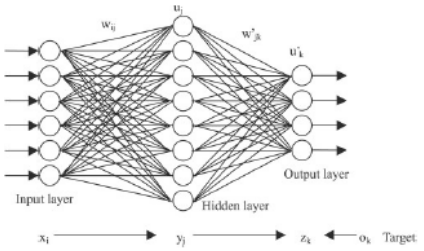


ls

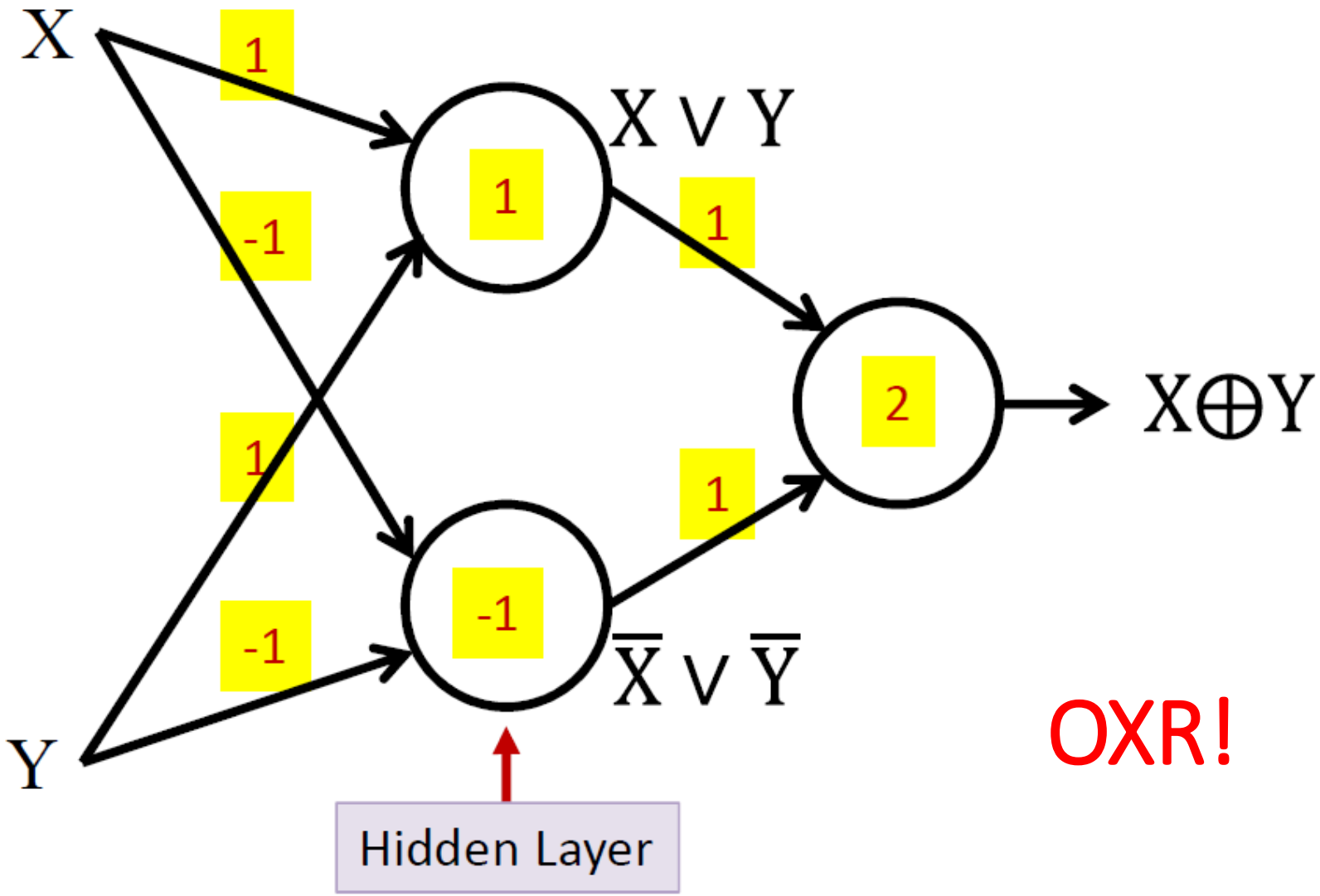
A single neuron is not enough





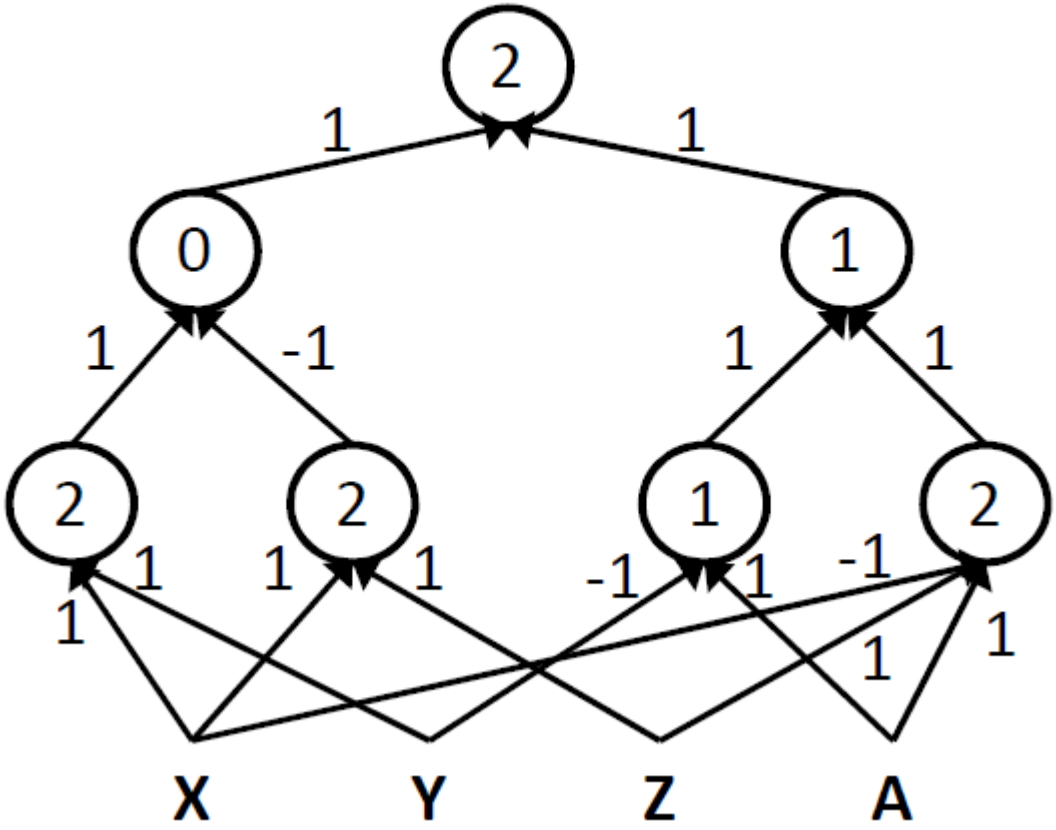


Multi-layer Perceptron (MLP)



Generic model

$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$

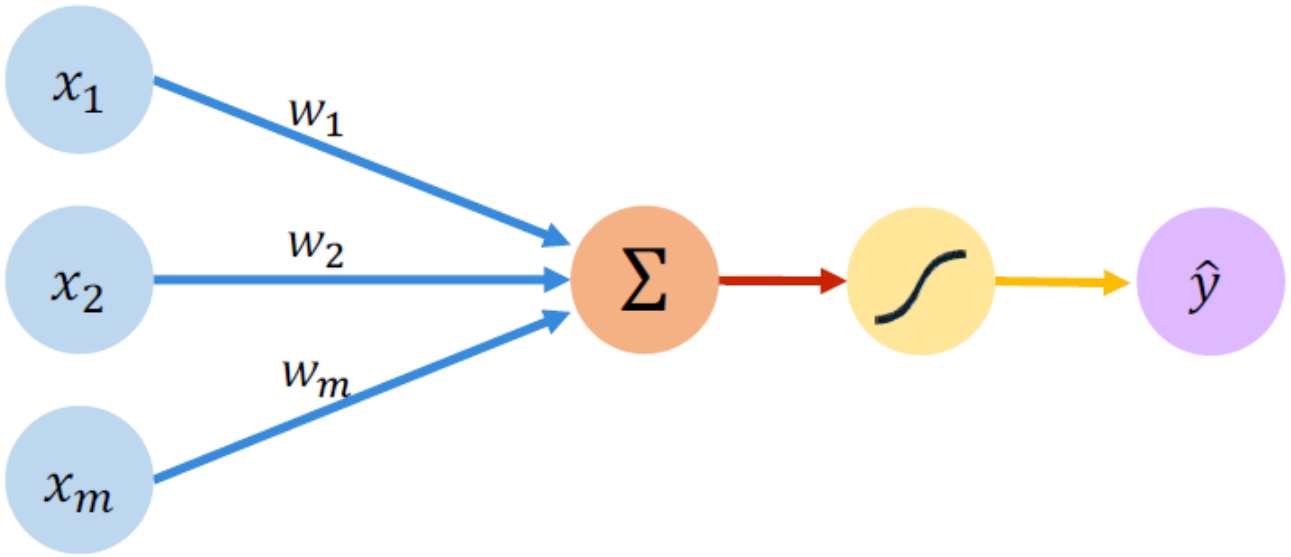


Perceptron

practice

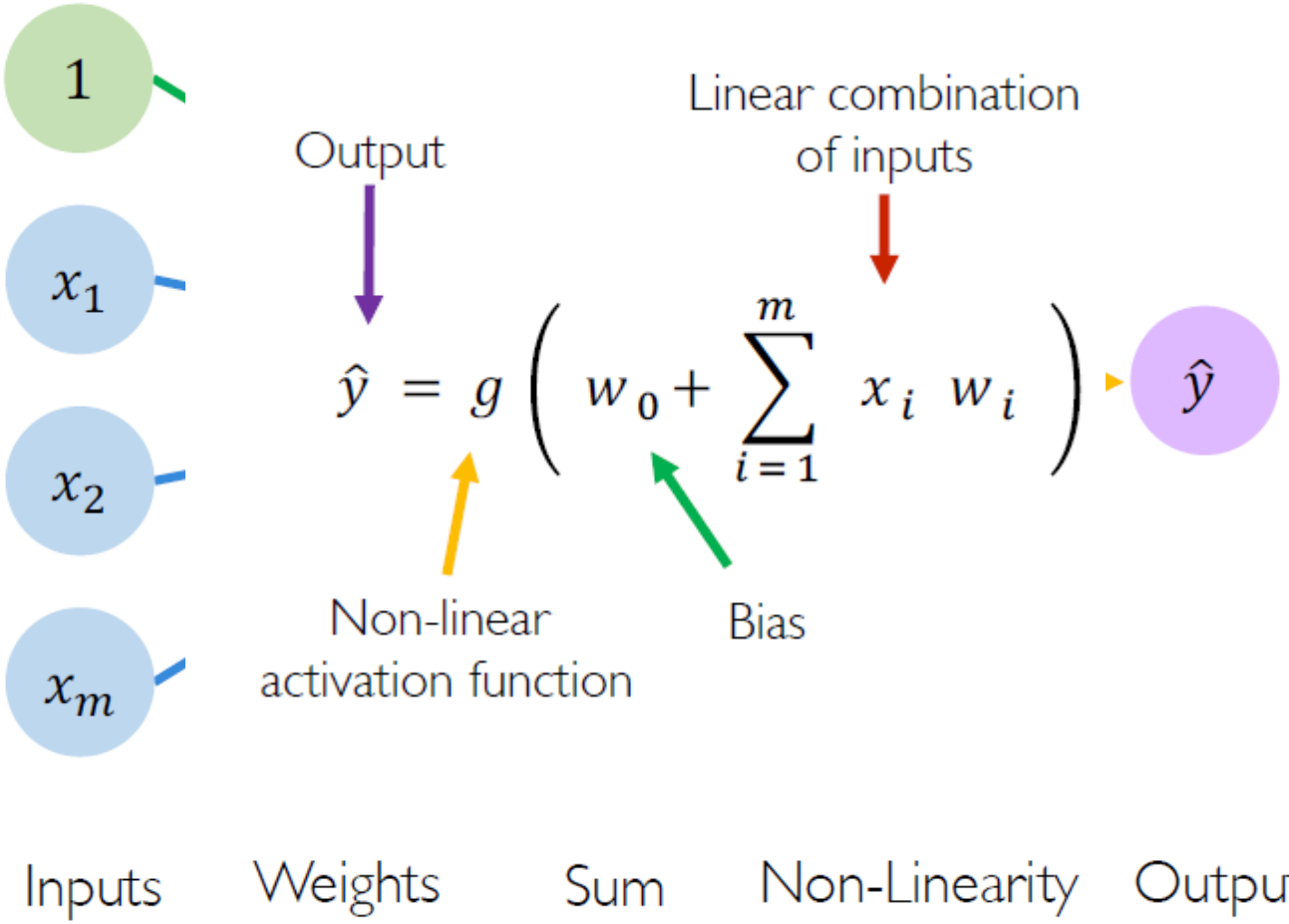
Perceptron: real data

A single net: Linear combination + Activations



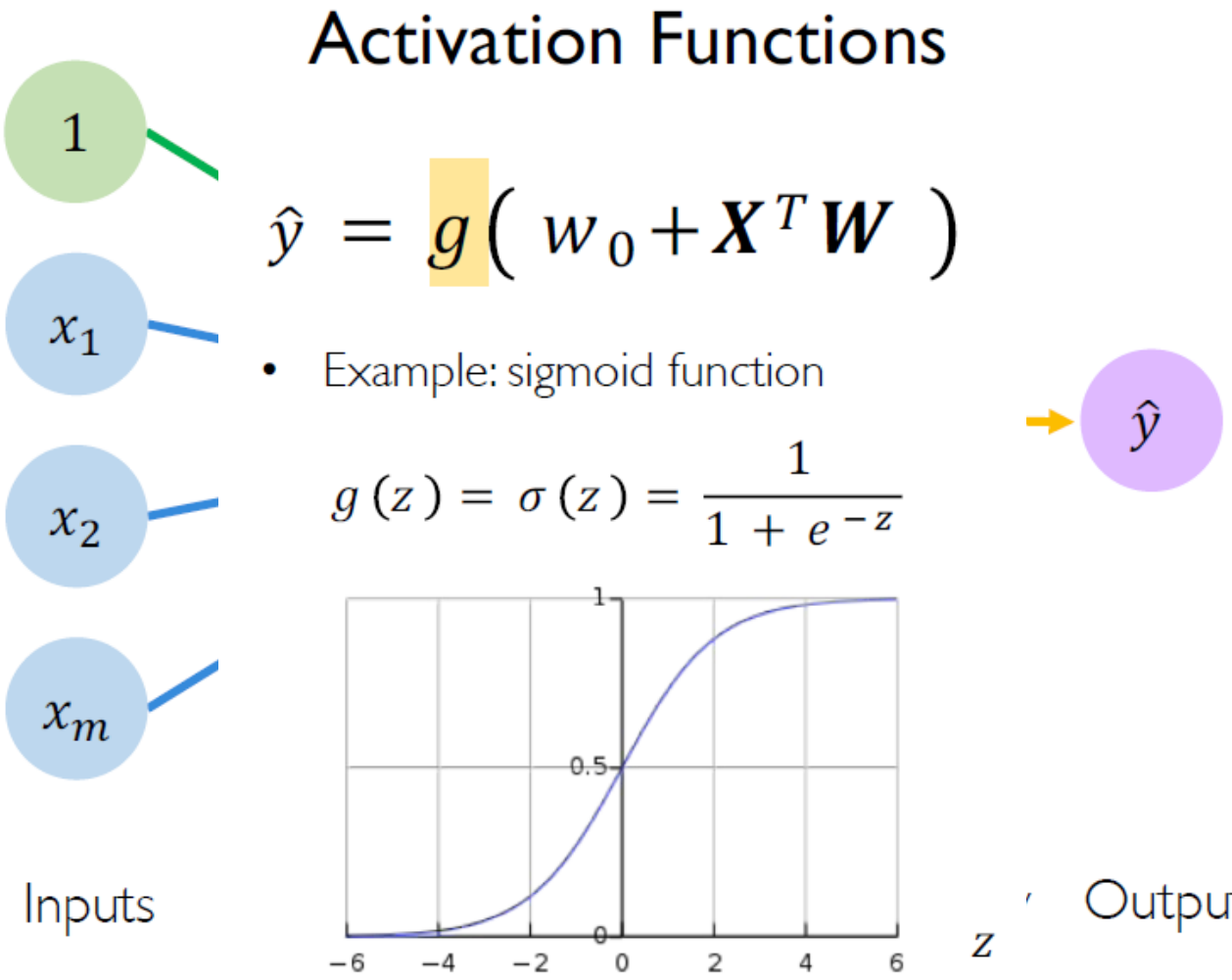
Perceptron: real data

A single net: Linear combination + Thresholding



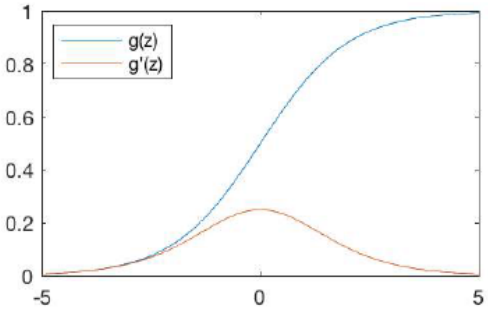
Perceptron: real data

A single net: Linear combination + Thresholding



Common activation functions

Sigmoid Function

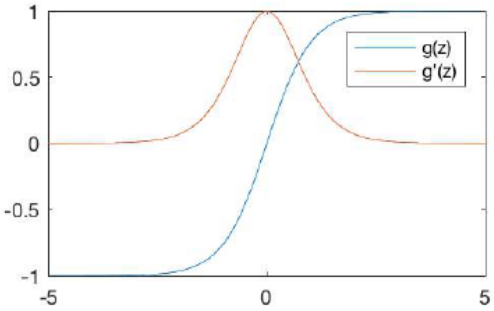


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

tf.nn.sigmoid(z)

Hyperbolic Tangent

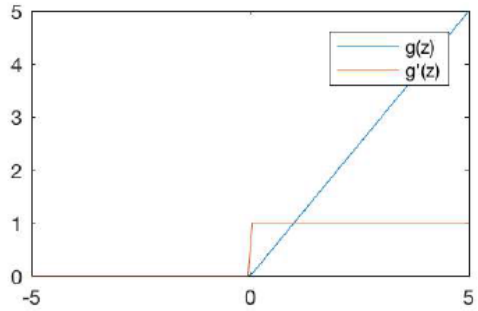


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

tf.nn.tanh(z)

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

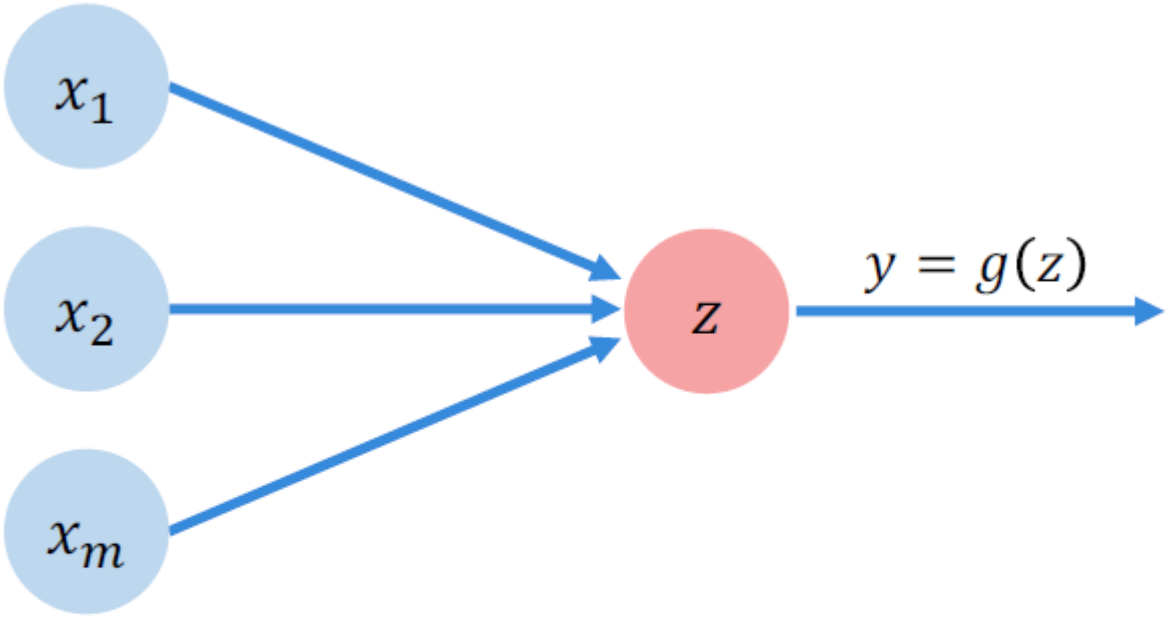
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

tf.nn.relu(z)

Building the network model

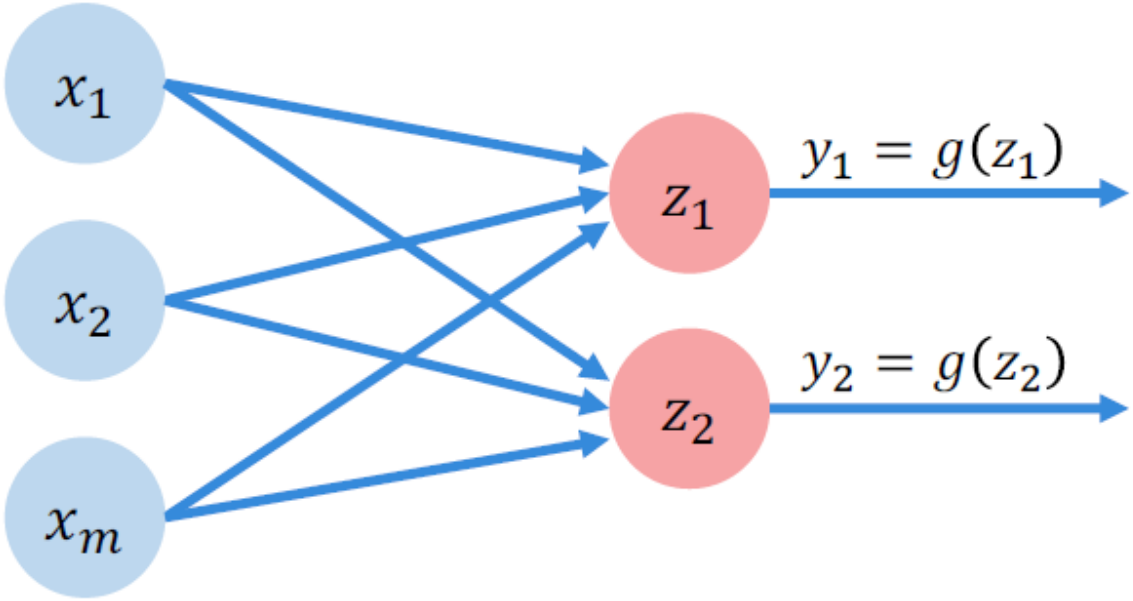
Toward MLP

Perceptron



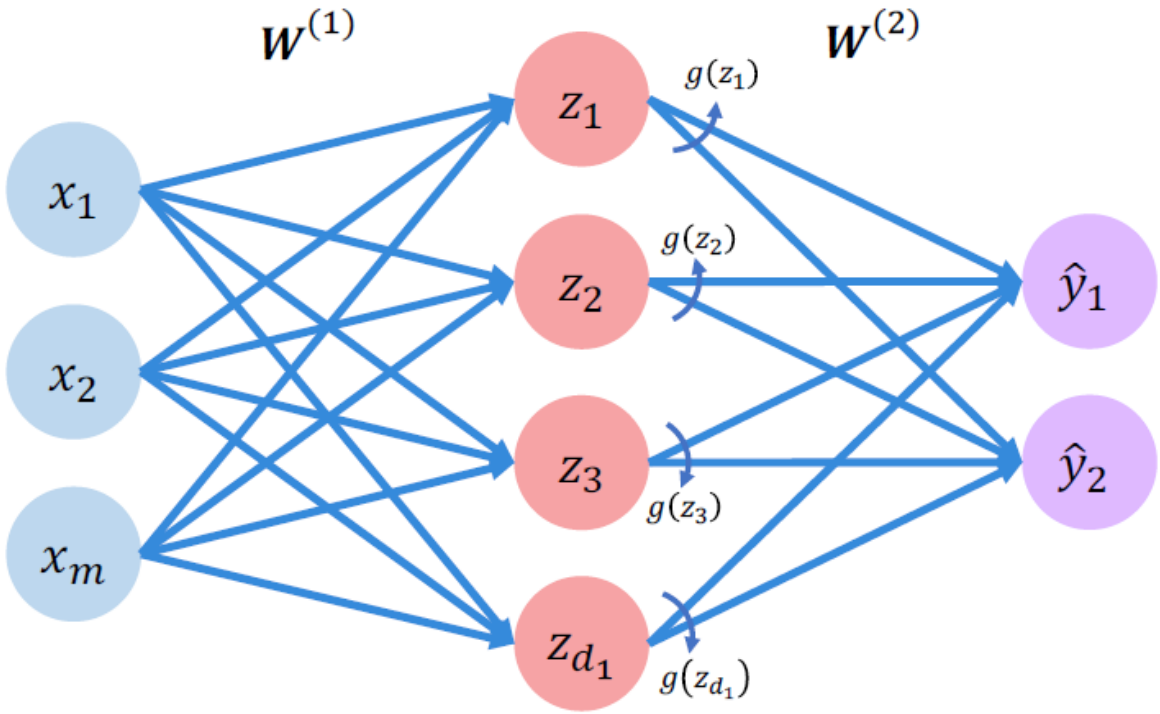
$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

Multi-output perceptron



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

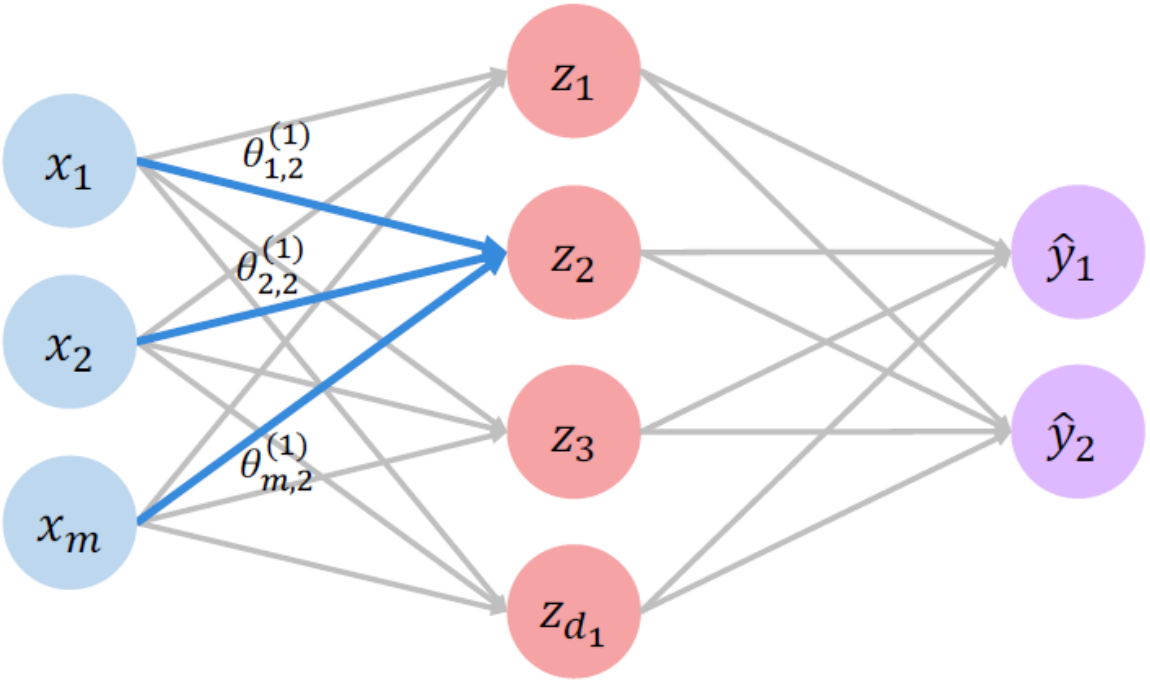
Single-layer Neural Net



Inputs Hidden Final Output

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)} \right)$$

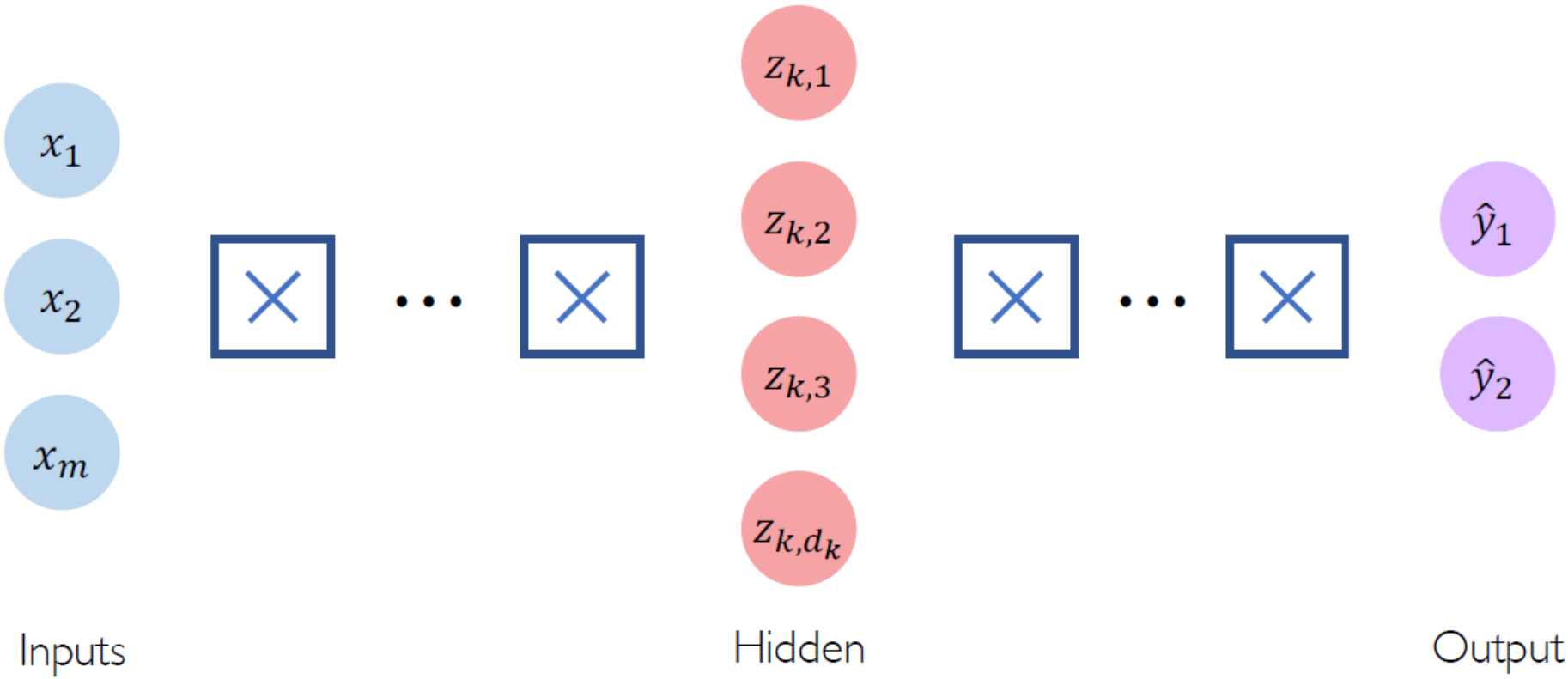
Single-layer Neural Net



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Deep Neural Network

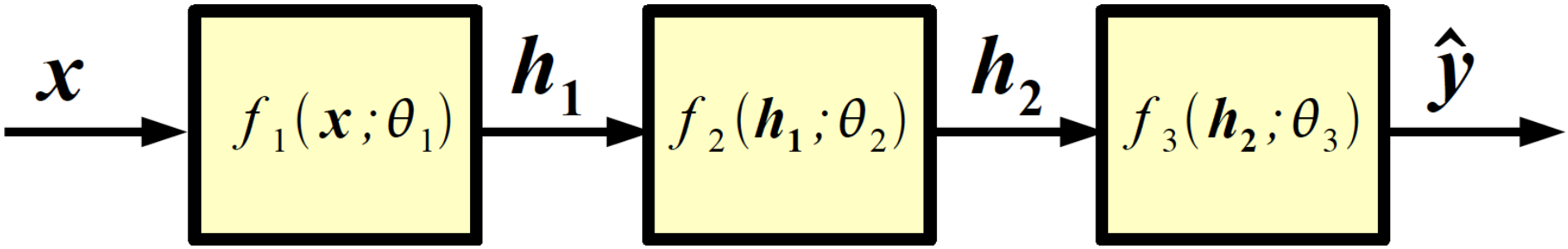


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

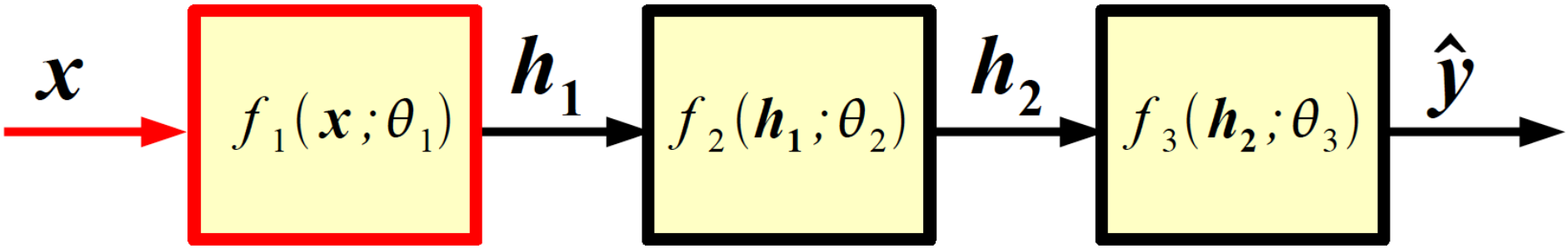
Forward propagation and Backprop.

Training the DNN

Neural nets

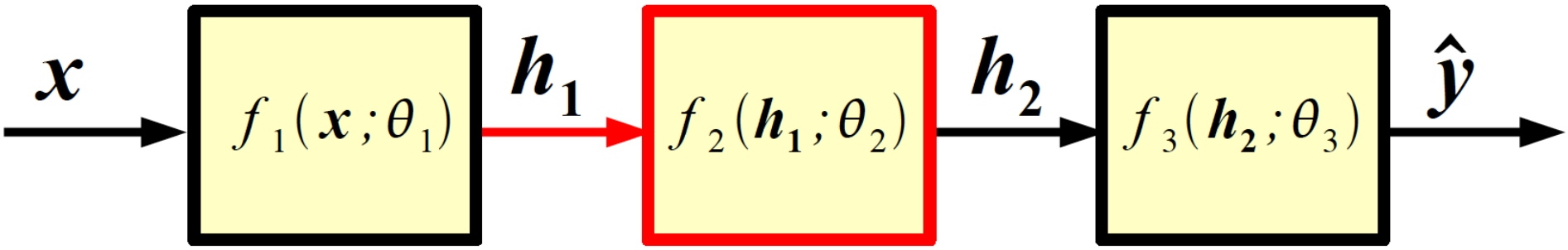

$\boldsymbol{x}$
$f_1(\boldsymbol{x};\theta_1)$
$\boldsymbol{h_1}$
$f_2(\boldsymbol{h_1};\theta_2)$
$\boldsymbol{h_2}$
$f_3(\boldsymbol{h_2};\theta_3)$
$\hat{\boldsymbol{y}}$

Forward propagation



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

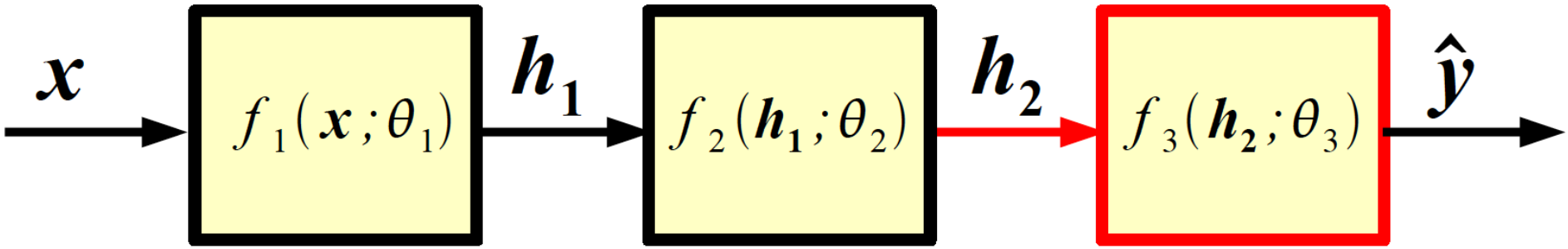
Forward propagation



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h_1};\theta_2)$

Forward propagation



1) Given x compute: $h_1 = f_1(x;\theta_1)$

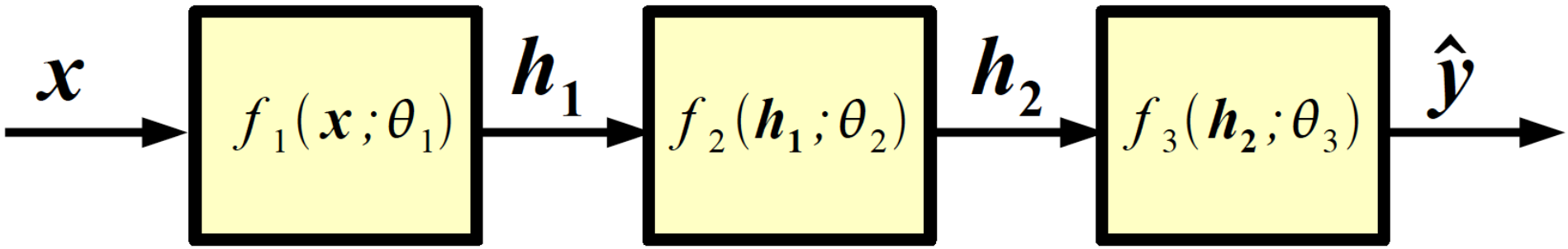
2) Given h_1 compute: $h_2 = f_2(h_1;\theta_2)$

3) Given h_2 compute: $\hat{y} = f_3(h_2;\theta_3)$

For instance,

$$\hat{y}_i = p(class = i|x) = \frac{e^{W_{3i}h_2 + b_{3i}}}{\sum_k e^{W_{3k}h_2 + b_{3k}}}$$

Forward propagation



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

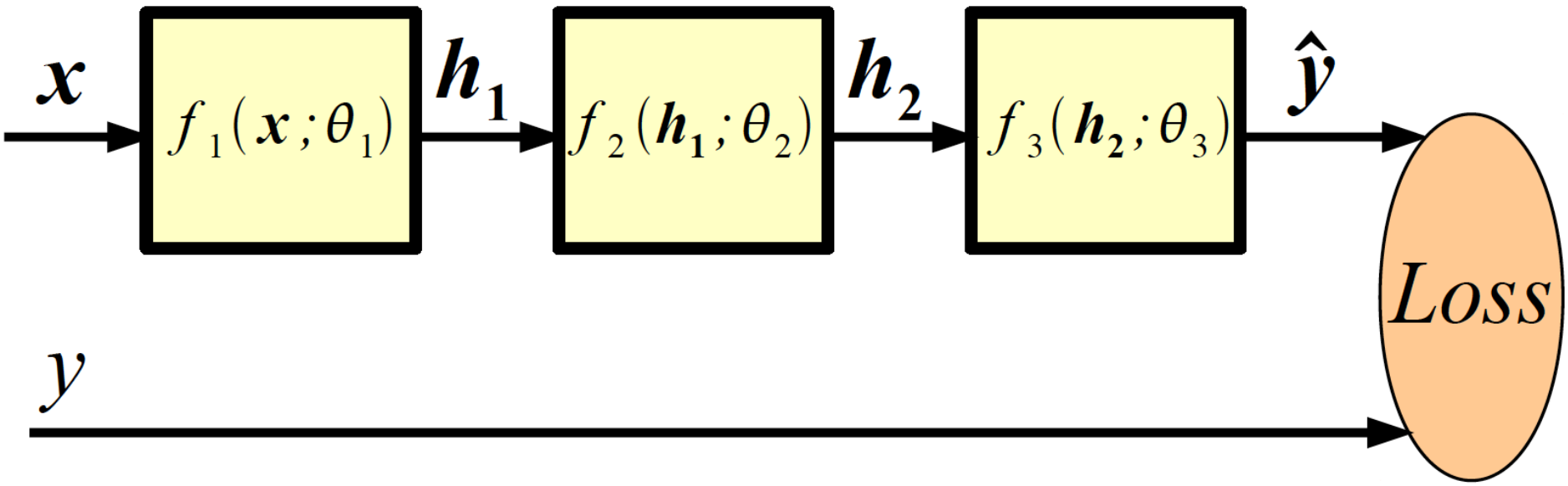
2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h_1};\theta_2)$

3) Given $\boldsymbol{h_2}$ compute: $\hat{\boldsymbol{y}} = f_3(\boldsymbol{h_2};\theta_3)$

This is the typical processing at test time.

At training time, we need to compute an error measure and tune the parameters to decrease the error.

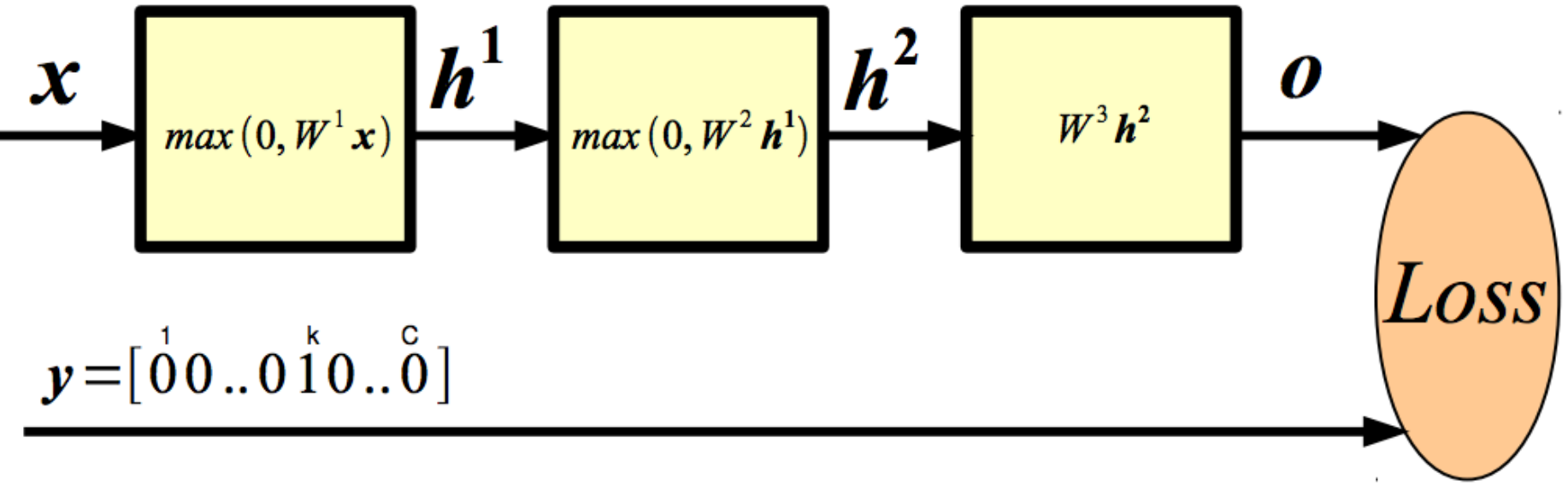
Loss



The measure of how well the model fits the training set is given by a suitable loss function: $L(\boldsymbol{x}, y; \boldsymbol{\theta})$

The loss depends on the input $\boldsymbol{x}$, the target label y, and the parameters $\boldsymbol{\theta}$.

How good a network?



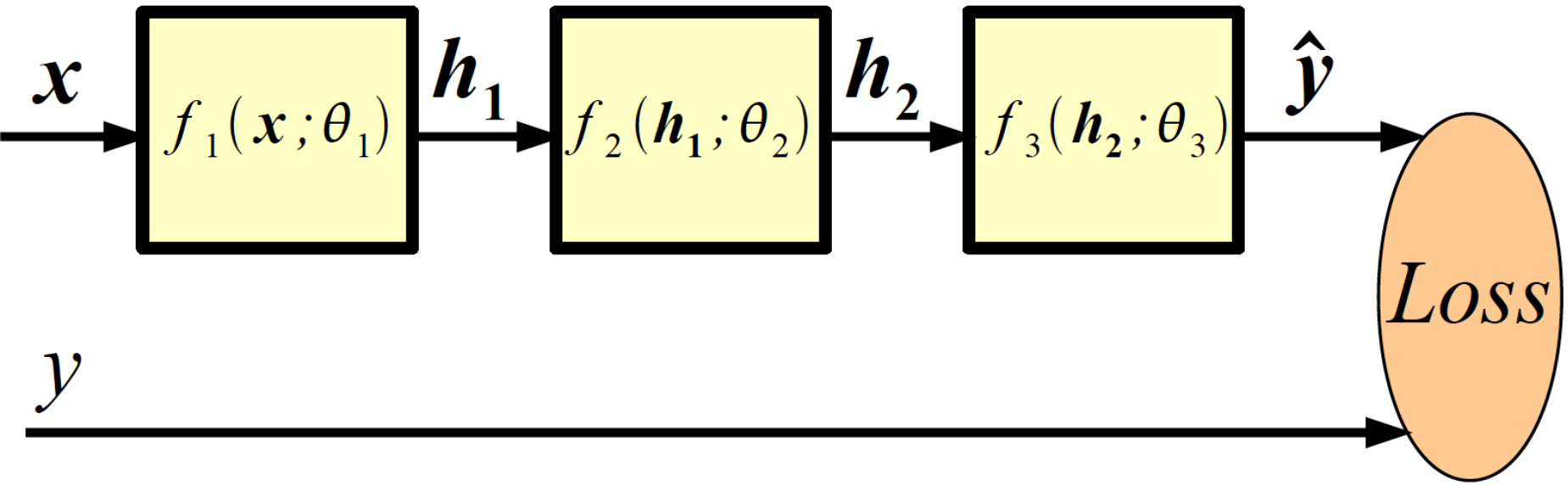
Probability of class k given input (softmax):

$$p(c_k = 1 | \boldsymbol{x}) = \frac{e^{o_k}}{\sum_{j=1}^{C} e^{o_j}}$$

(Per-sample) **Loss**; e.g., negative log-likelihood (good for classification of small number of classes):

$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j | \boldsymbol{x})$$

Loss



Q.: how to tune the parameters to decrease the loss?

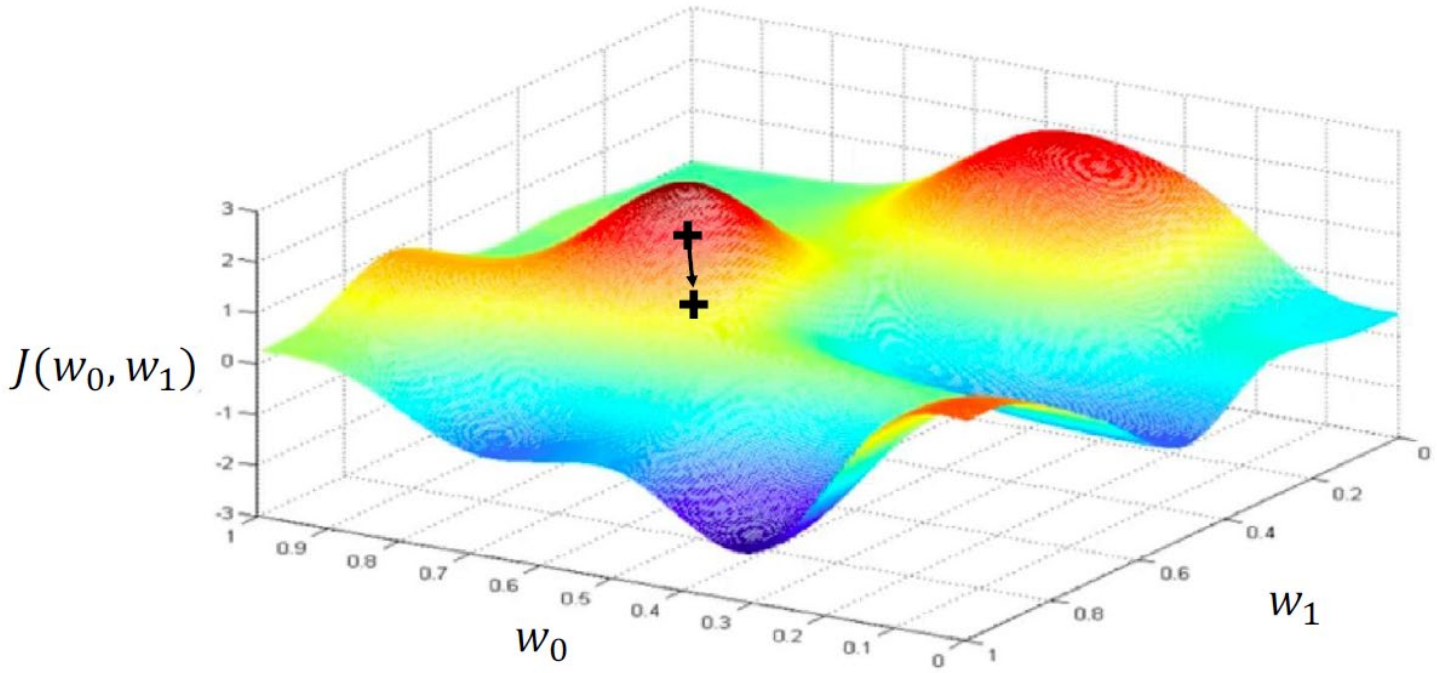
If loss is (a.e.) differentiable we can compute gradients.

We can use chain-rule, a.k.a. **back-propagation**, to compute the gradients w.r.t. parameters at the lower layers.

Rumelhart et al. "Learning internal representations by back-propagating.." Nature 1986

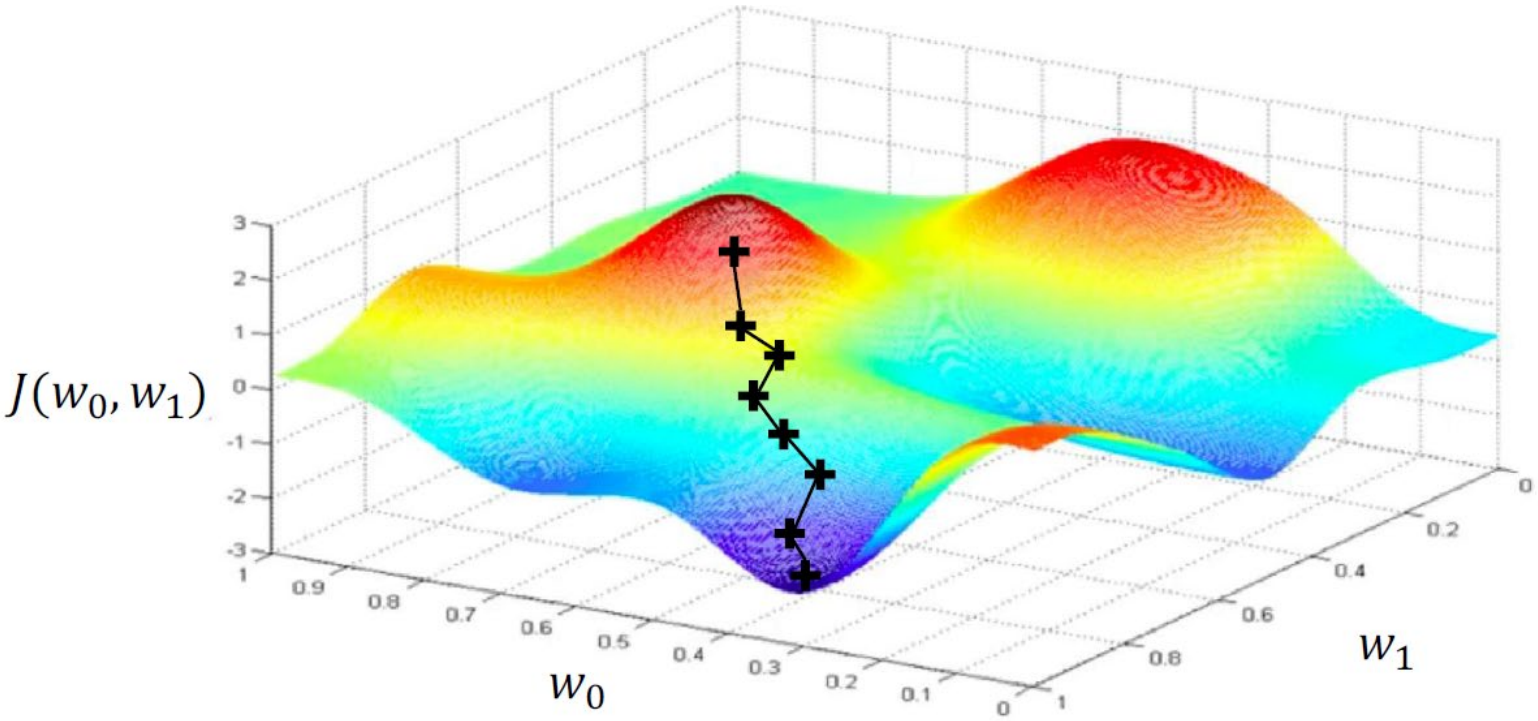
Loss minimization

Move somewhere in opposite direction of gradient.



Gradient descent

Repeat until convergence.

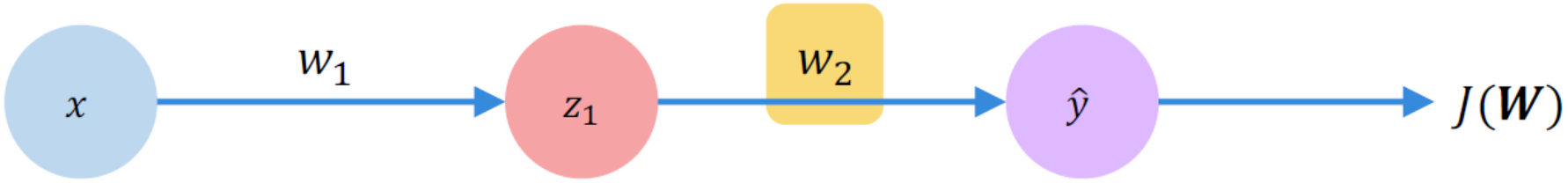


Gradient descent

Algorithm

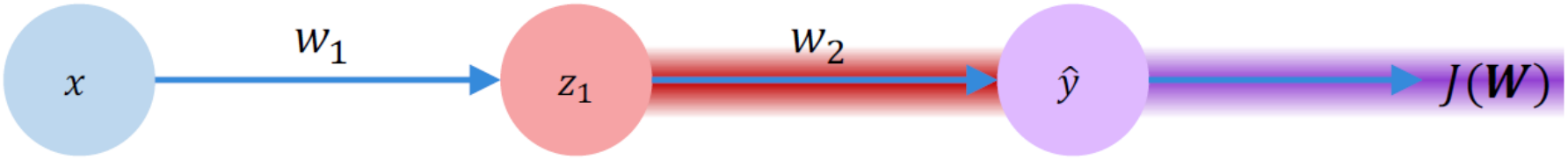
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Backpropagation: Gradient



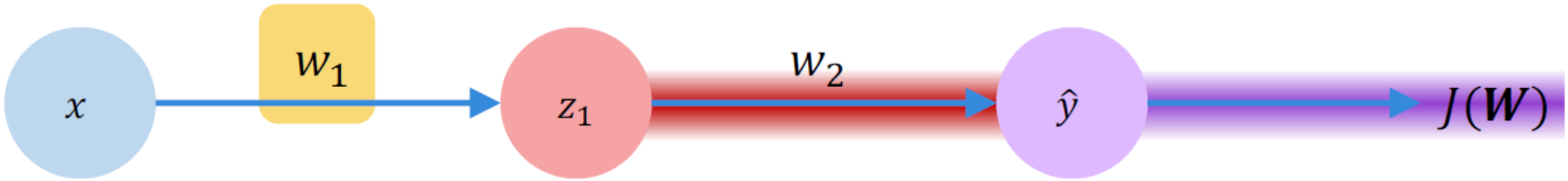
$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

Backpropagation: Gradient



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Backpropagation: Gradient

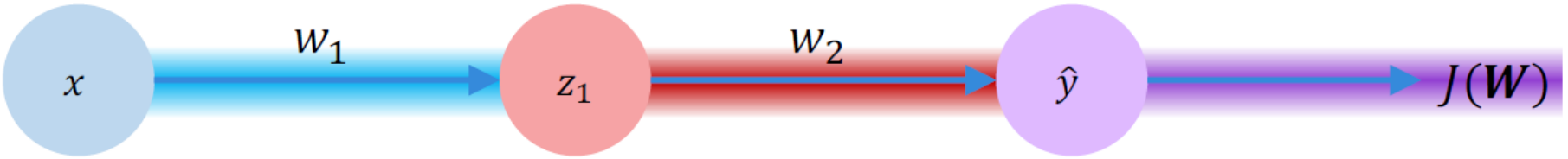


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

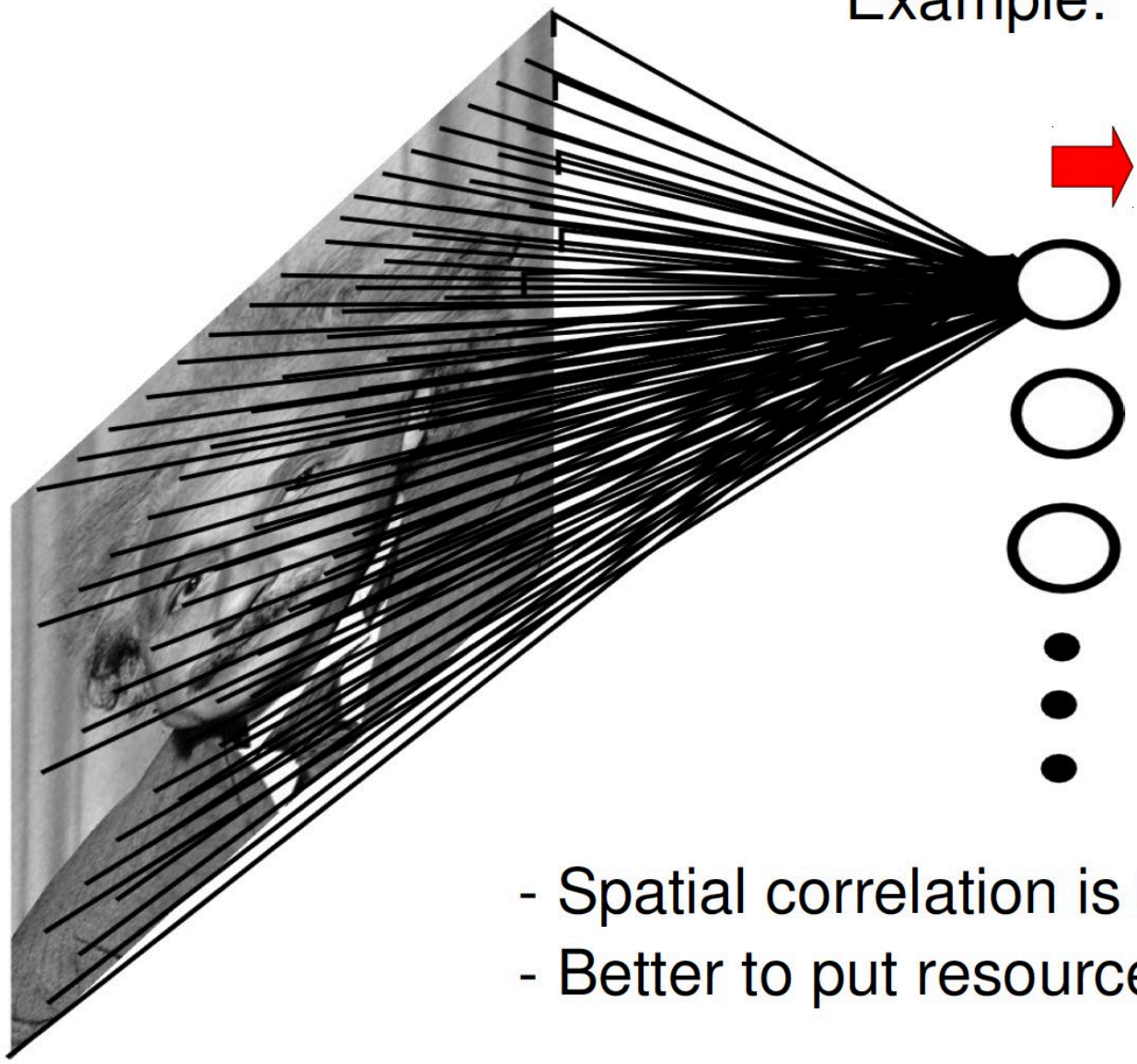
Backpropagation: Gradient



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Convolutional neural nets

Fully connected neural net



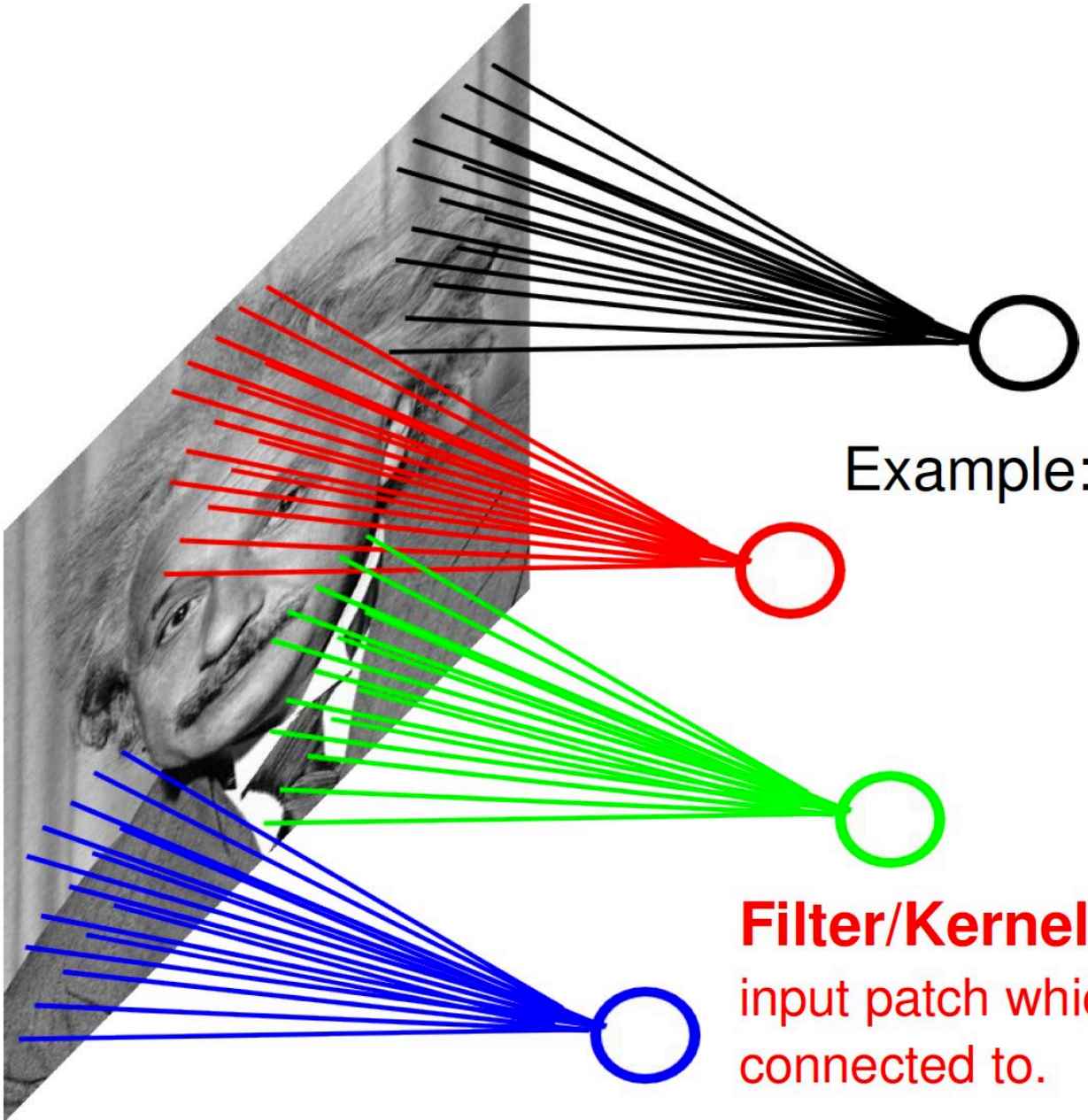
Example: 1000x1000 image

1M hidden units

10^12 parameters!!!

- Spatial correlation is local
- Better to put resources elsewhere!

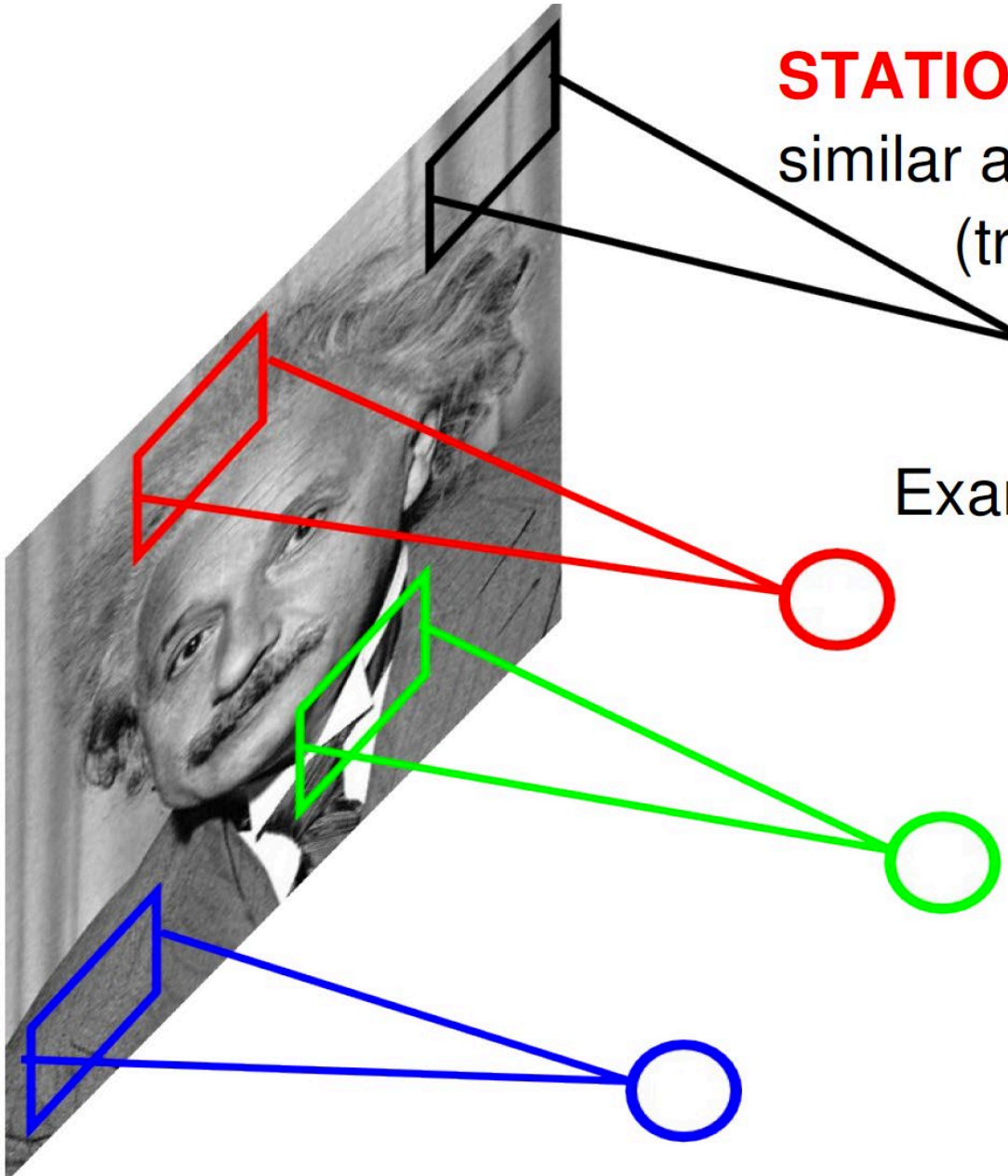
Locally connected neural net



Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Filter/Kernel/Receptive field:
input patch which the hidden unit is connected to.

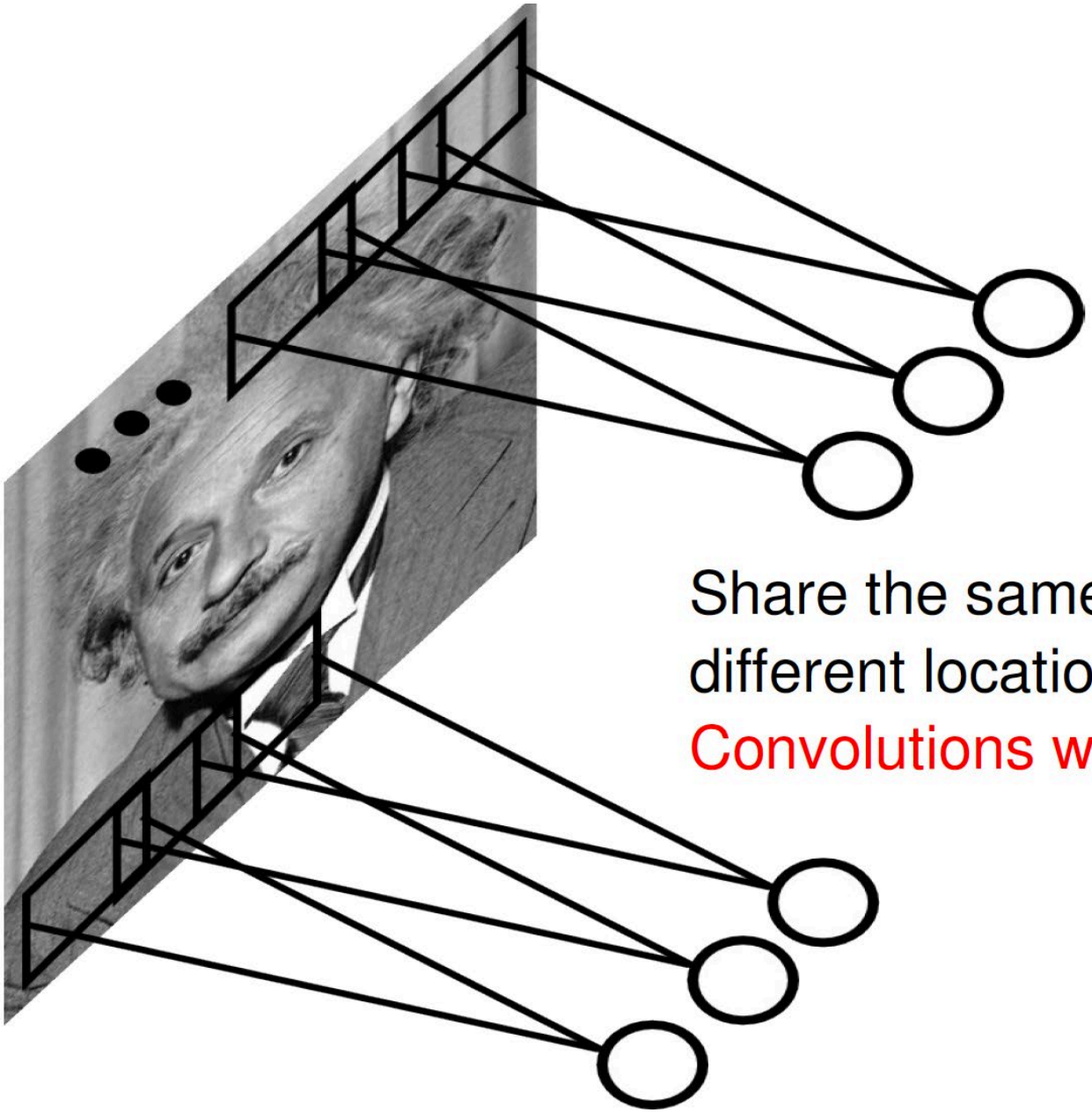
Locally connected neural net



STATIONARITY? Statistics are similar at different locations (translation invariance)

Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

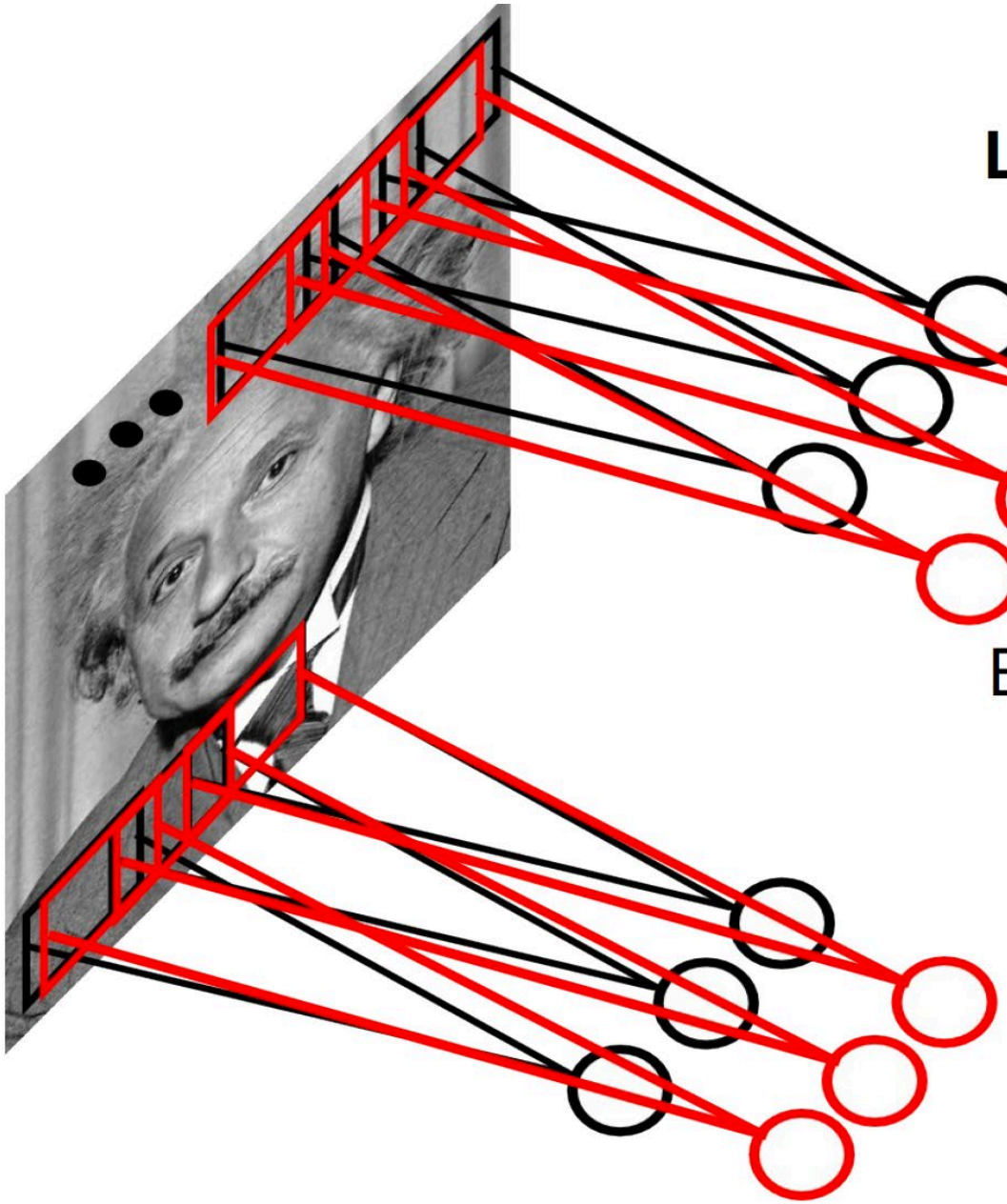
Convolutional net



Share the same parameters across different locations:

Convolutions with learned kernels

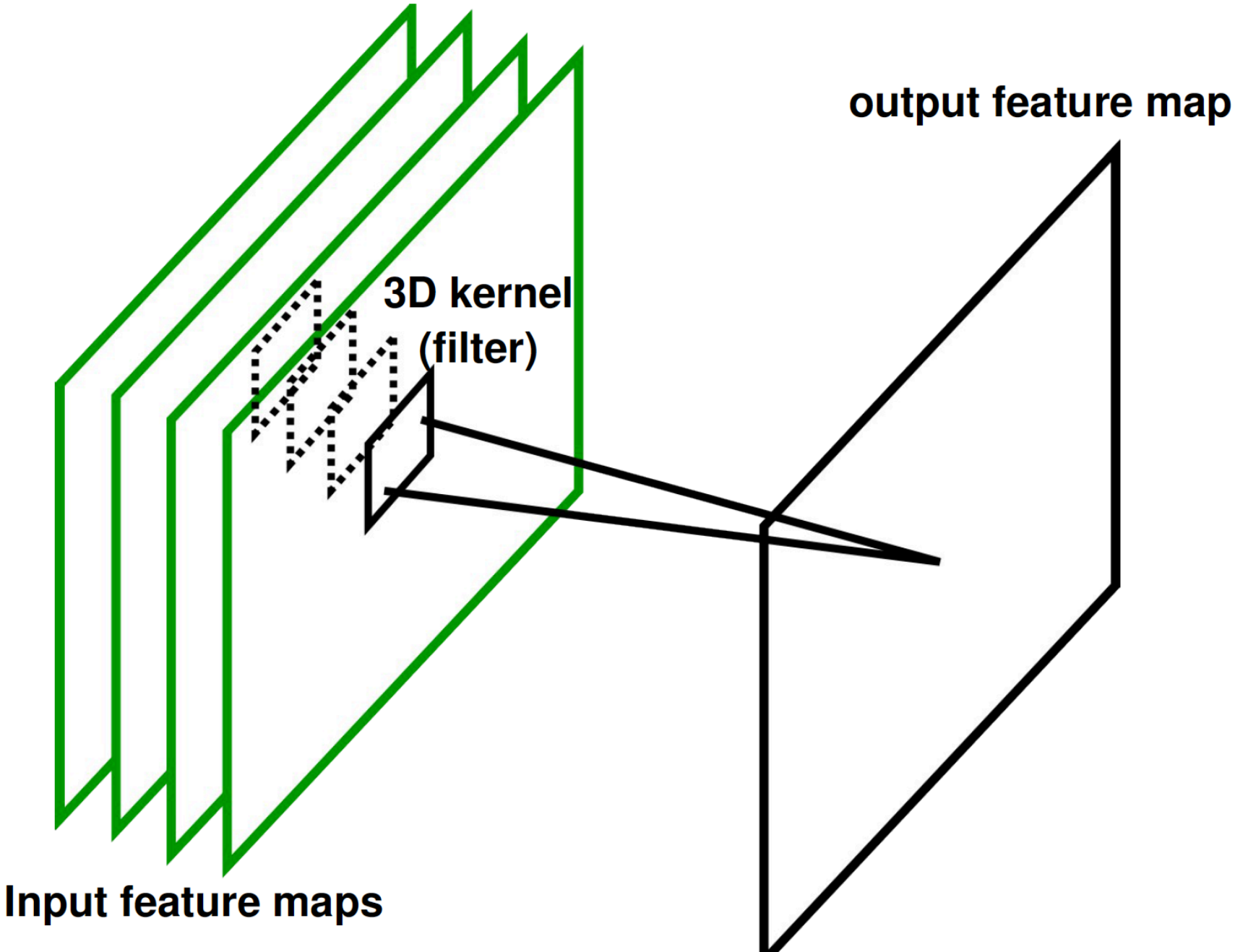
Convolutional net



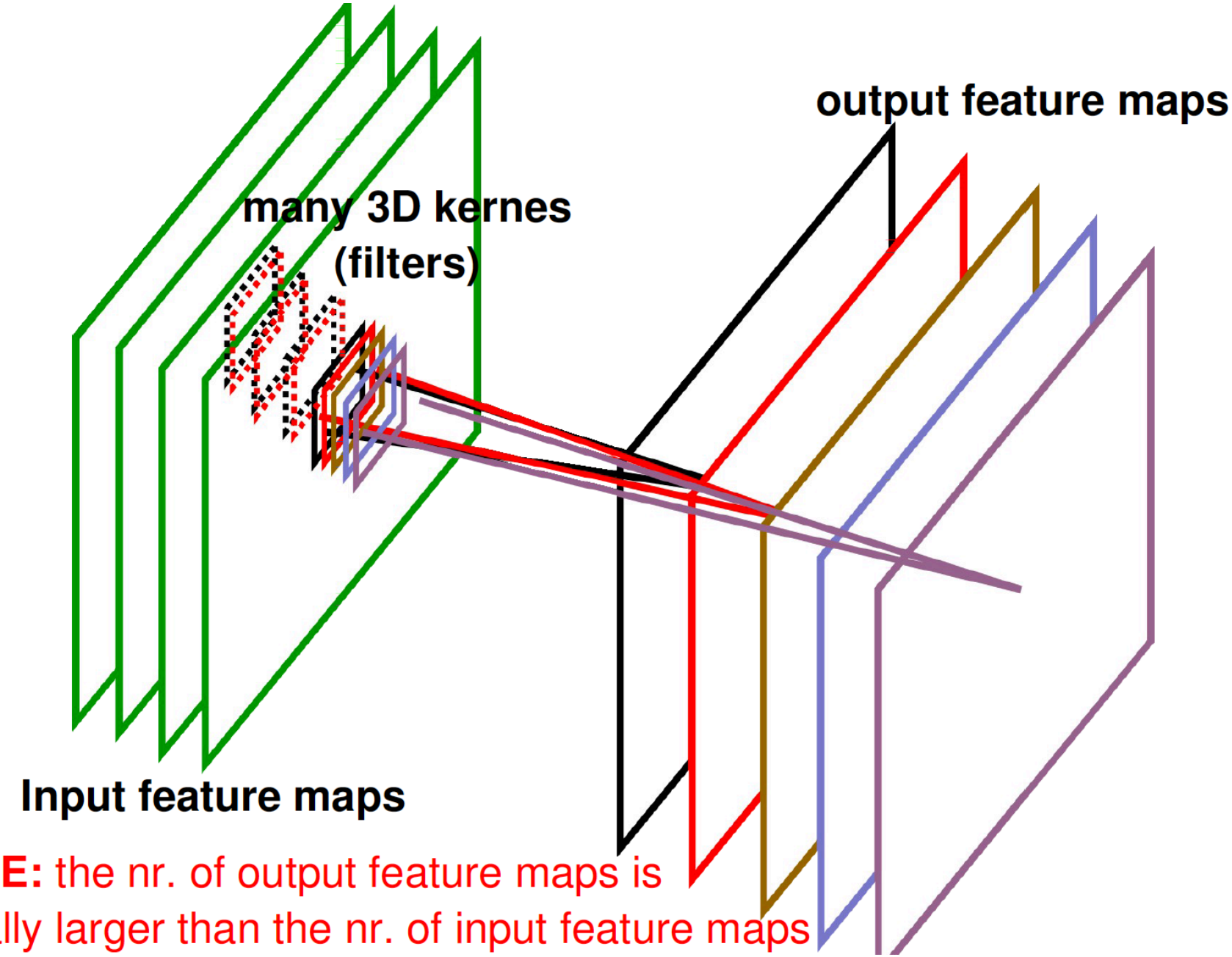
Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

Convolutional layer

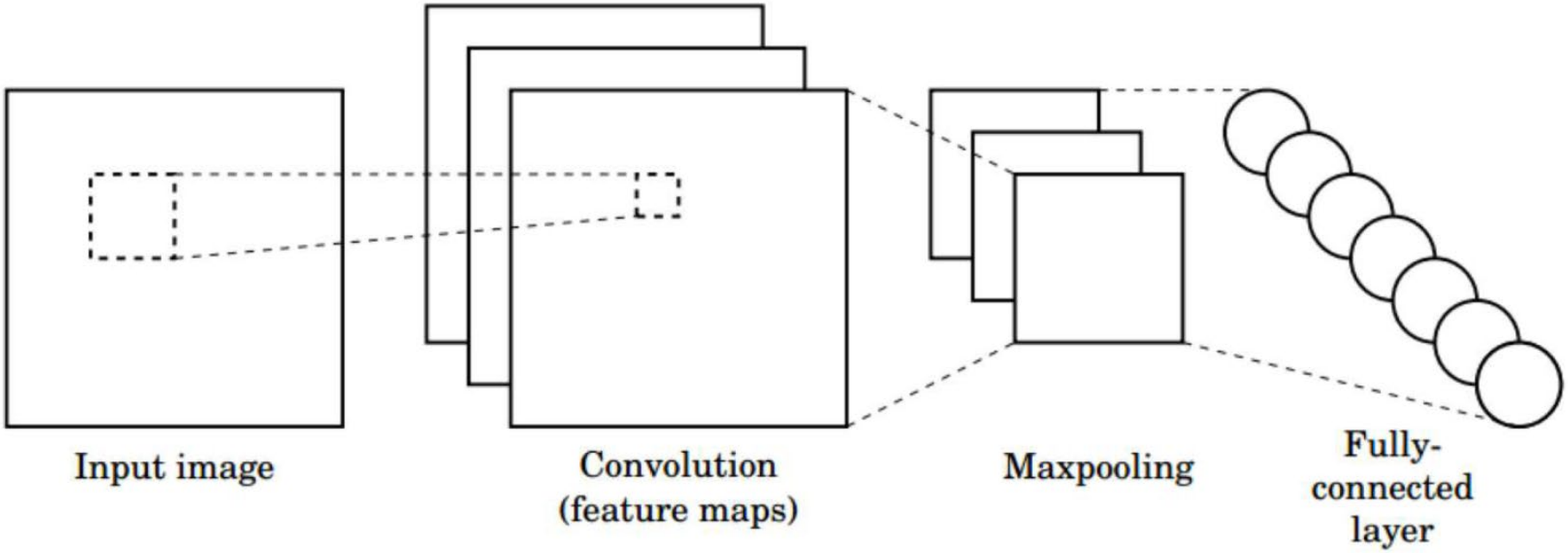


Convolutional layer



NOTE: the nr. of output feature maps is usually larger than the nr. of input feature maps

Convolutional neural nets

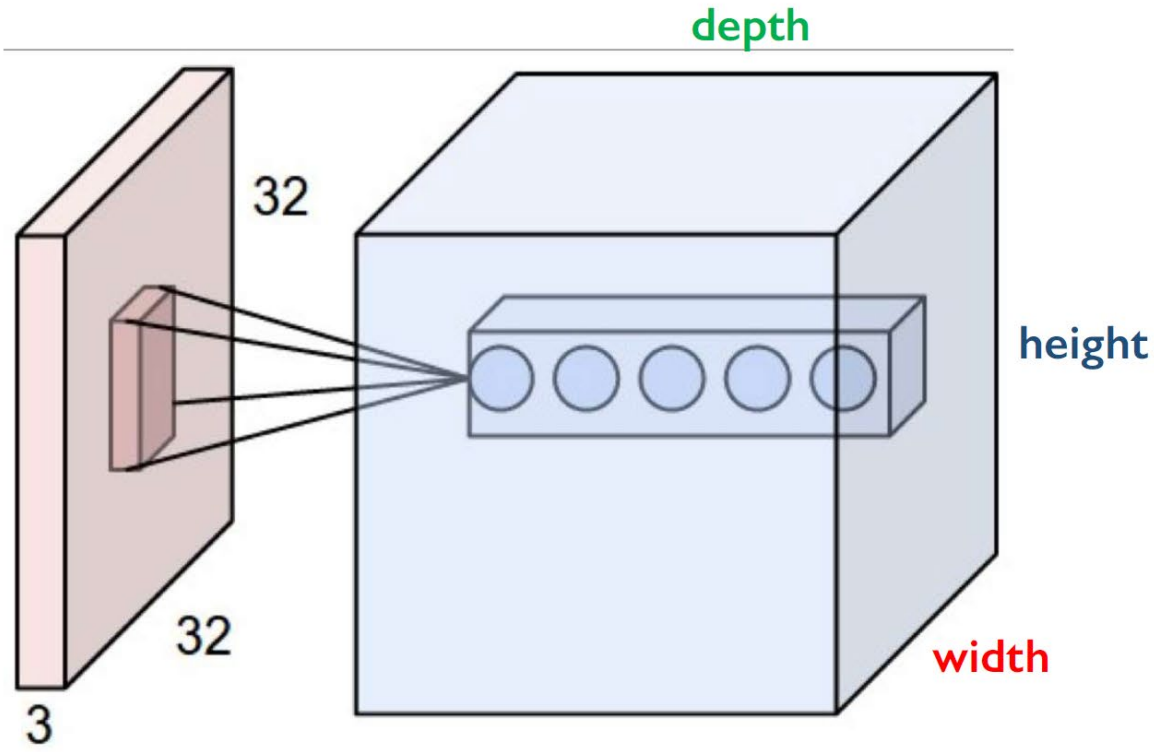


Convolution: Local associations by kernel

Non-linearity: ReLu

Pooling: Downsampling operation

CNN
: Feature map by convolution



Layer Dimensions:

$h \; x \; w \; x \; d$

where h and w are spatial dimensions
d (depth) = number of filters

Stride:

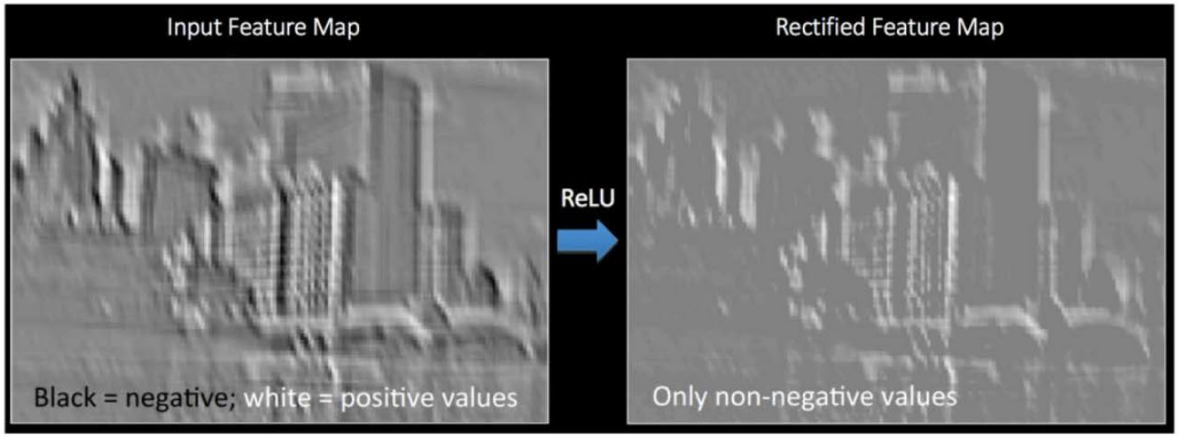
Filter step size

Receptive Field:

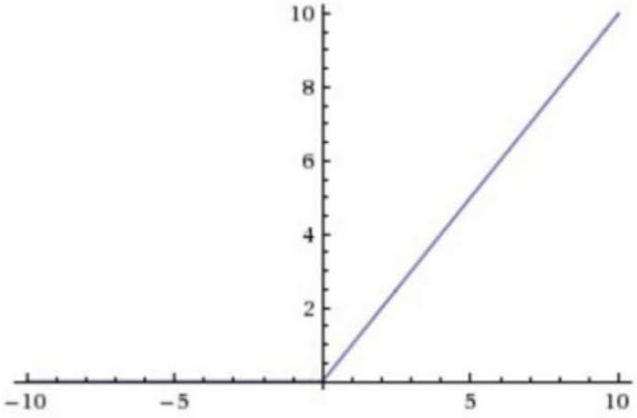
Locations in input image that
a node is path connected to

Non-linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**

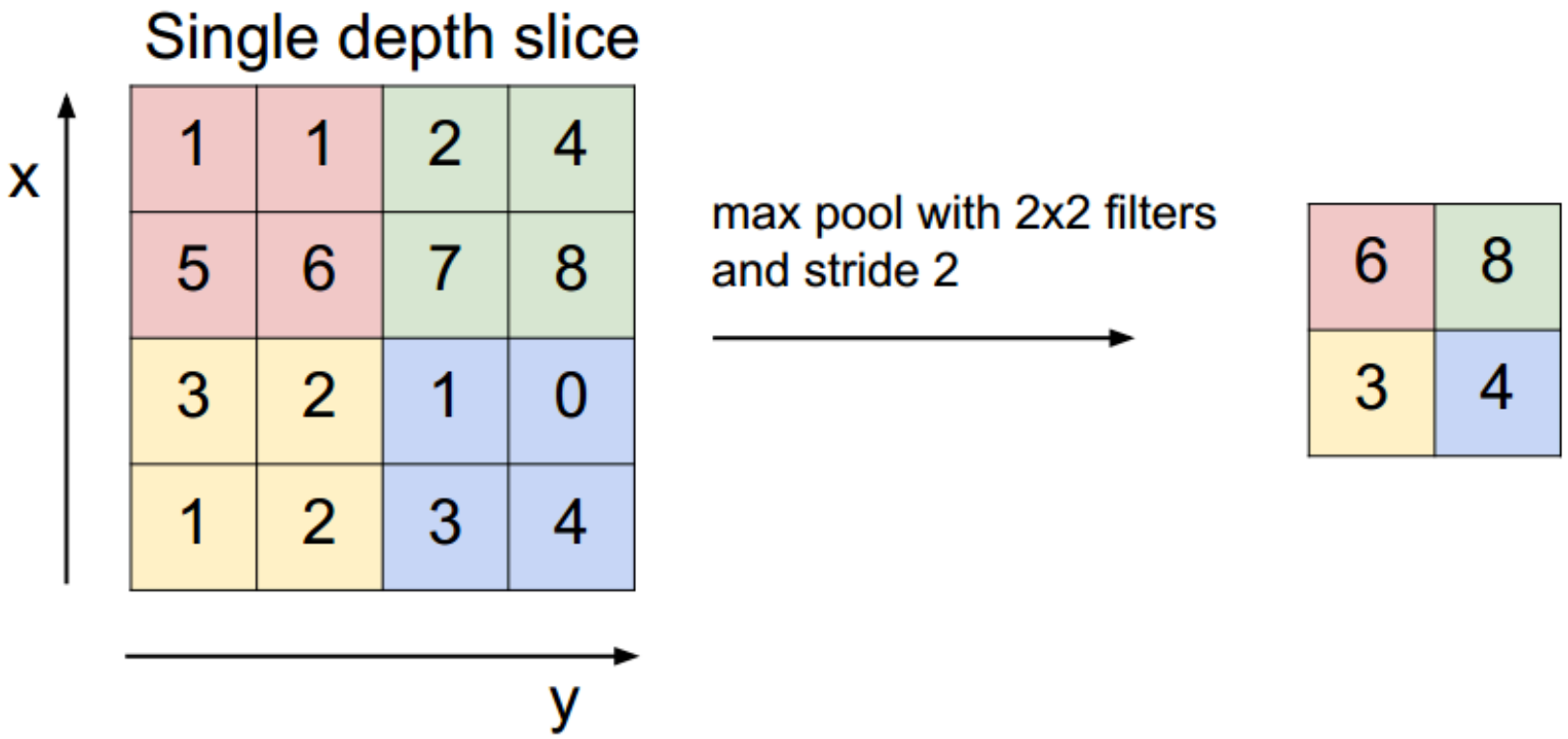


Rectified Linear Unit (ReLU)



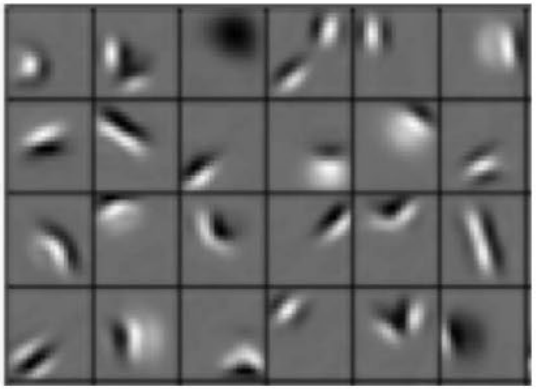
$$g(z) = \max(0, z)$$

Max pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Representations learned by CNN

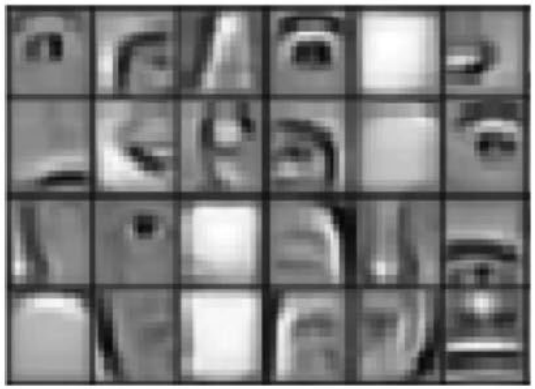
Low level features



Edges, dark spots

Conv Layer 1

Mid level features



Eyes, ears, nose

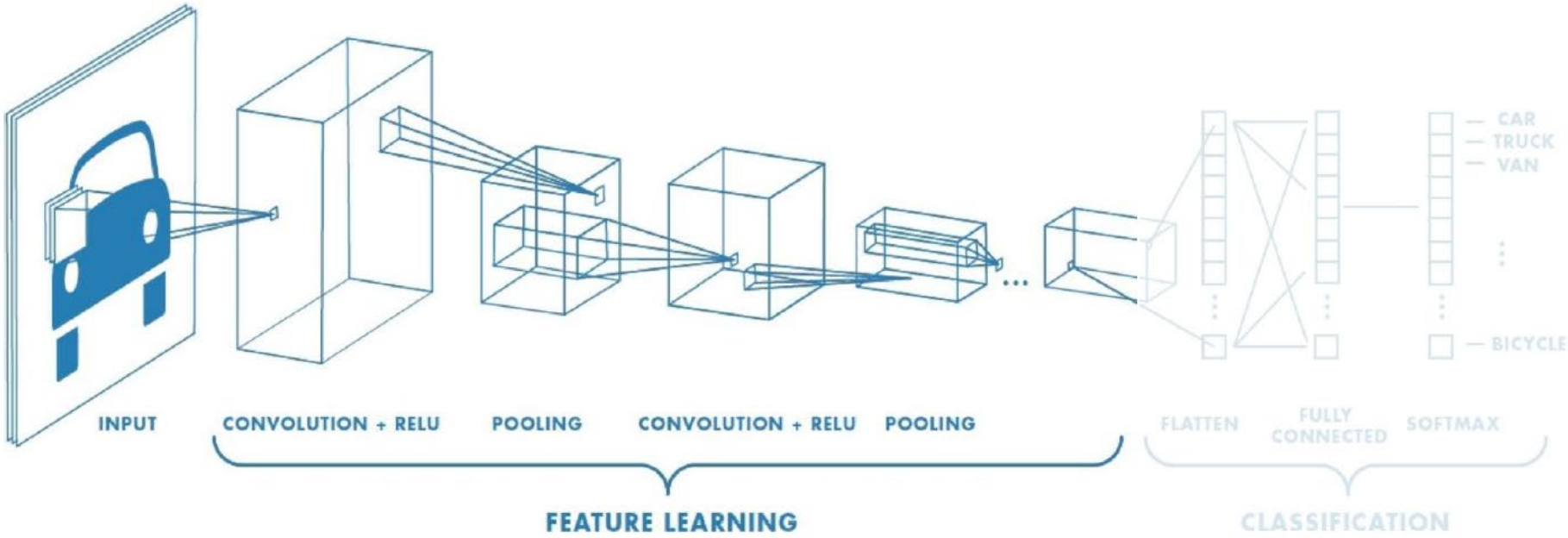
Conv Layer 2

High level features

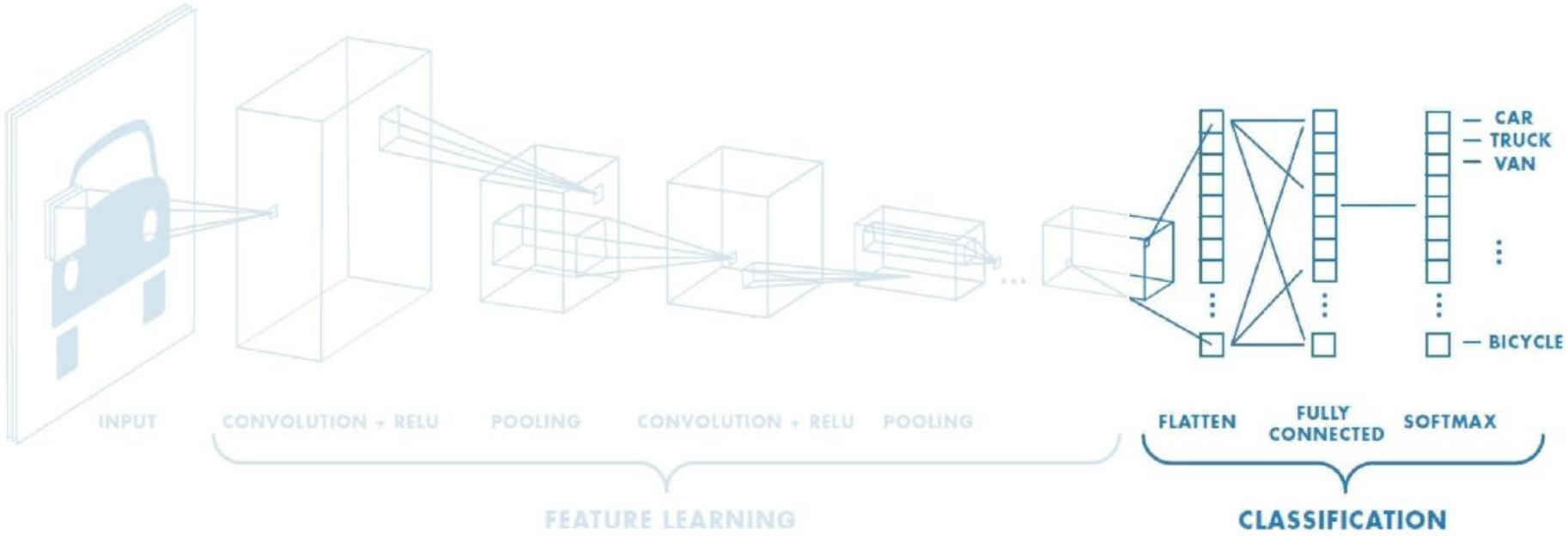
Facial structure

Conv Layer 3

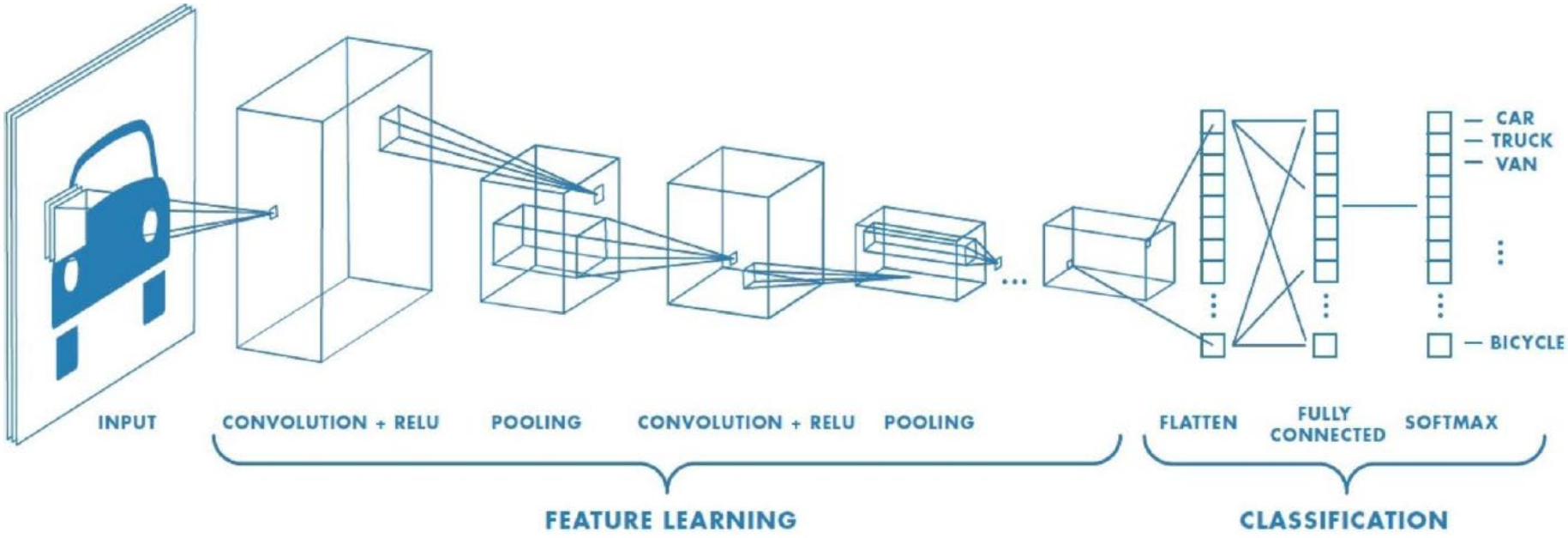
CNN for classification



CNN for classification



CNN for classification



Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\hat{\boldsymbol{y}}^{(i)}\right)$$