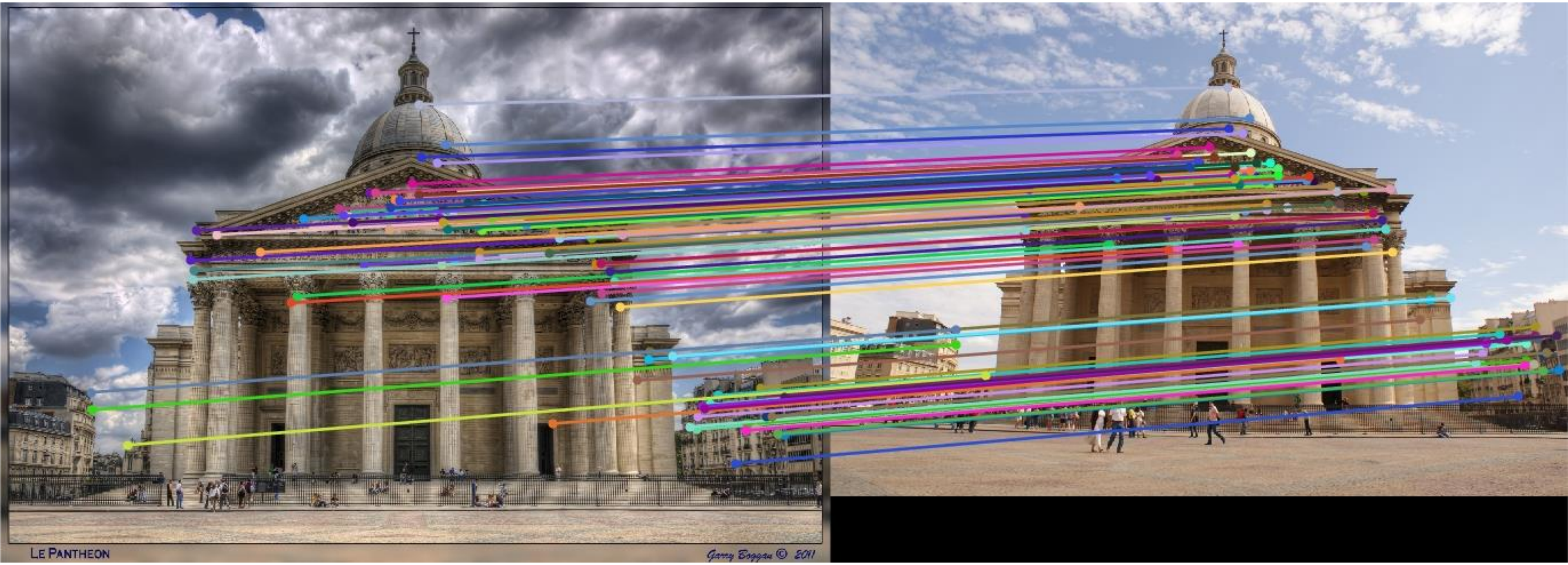
Project 2 highlights



Seunghwan Lee

Project 2 highlights

- Jared Duncan
- Nathaniel Glaser
- Anh Thai
- Yuhui Zhao
- Jiaye Liu
- seunghwan lee
- yinglin li

Deep Learning 1
Neural Net Basics

Computer Vision

James Hays

Many slides by Marc'Aurelio Ranzato

Outline

- Neural Networks
- *Convolutional* Neural Networks
- Variants
  - Detection
  - Segmentation
  - Siamese Networks
- Visualization of Deep Networks

Supervised Learning

$\{(\boldsymbol{x}^i, y^i), i=1 \ldots P\}$ training dataset

$\boldsymbol{x}^i$ i-th input training example

y^i i-th target label

P number of training examples

Goal: predict the target label of unseen inputs.

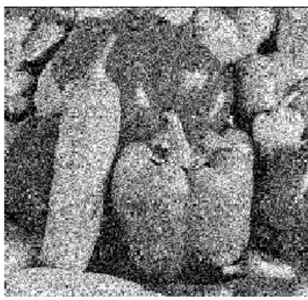
Supervised Learning: Examples

Classification

"dog"

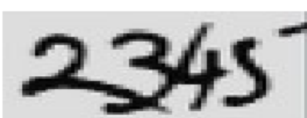
classification

Denoising



regression

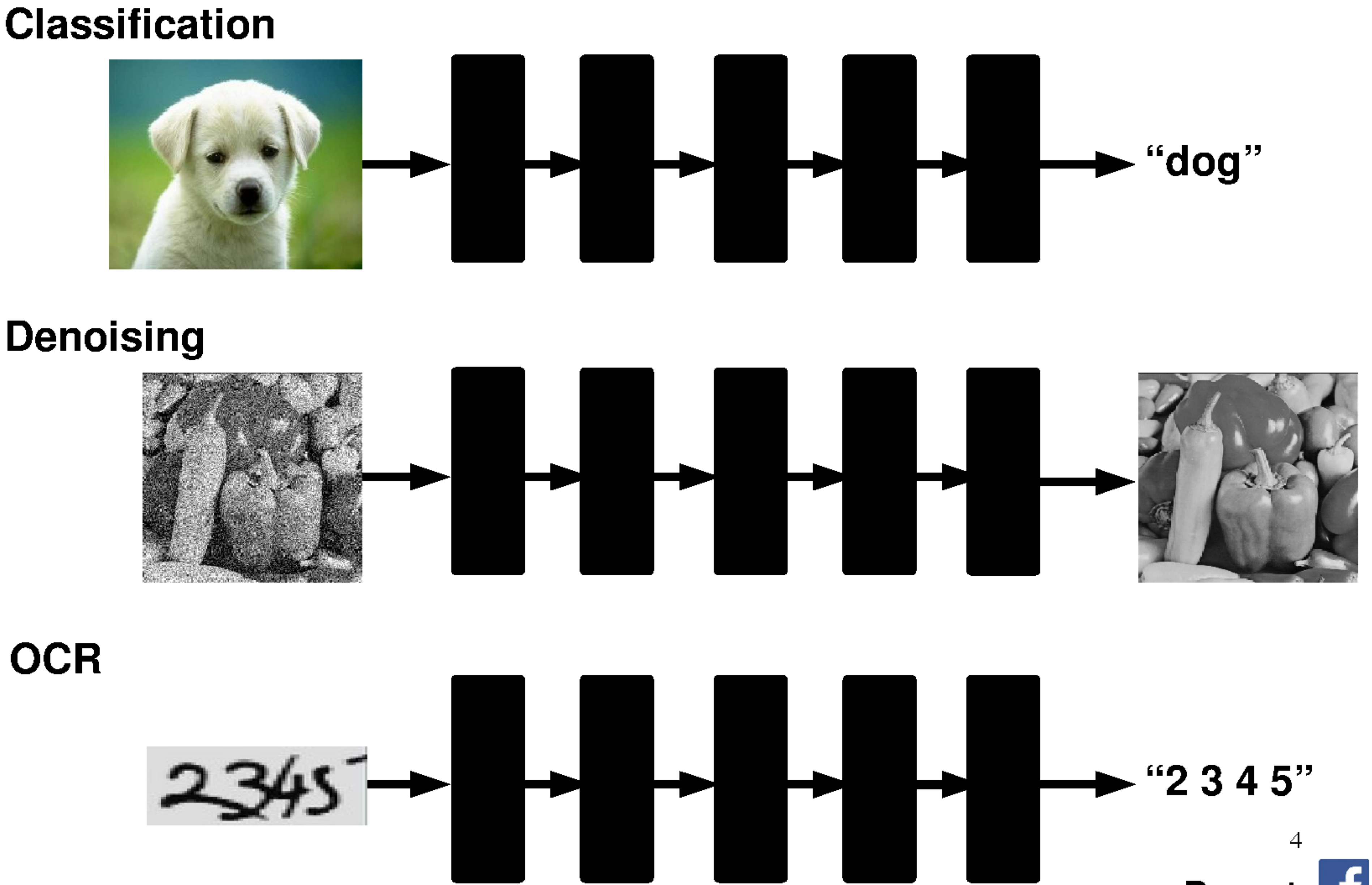
OCR



"2 3 4 5"

structured prediction

Supervised Deep Learning



Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips

Neural Networks

Assumptions (for the next few slides):

- The input image is vectorized (disregard the spatial layout of pixels)
- The target label is discrete (classification)

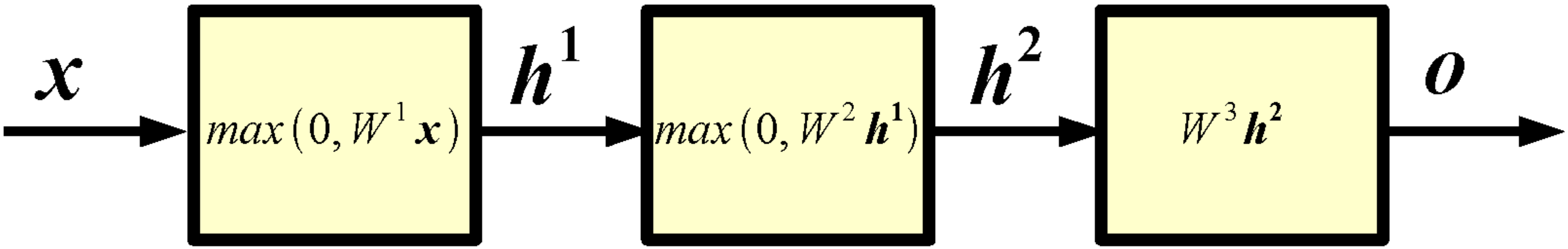
Question: what class of functions shall we consider to map the input into the output?

Answer: composition of simpler functions.

Follow-up questions: Why not a linear combination? What are the “simpler” functions? What is the interpretation?

Answer: later...

Neural Networks: example



$\boldsymbol{x}$ input

$\boldsymbol{h}^1$ 1-st layer hidden units

$\boldsymbol{h}^2$ 2-nd layer hidden units

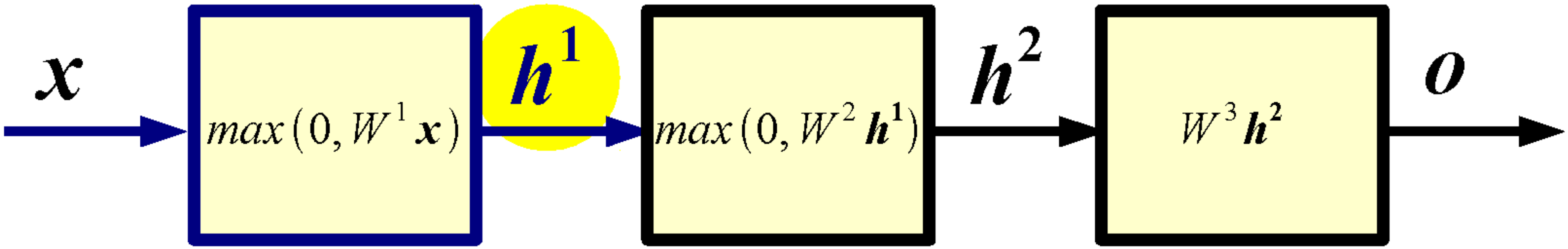
$\boldsymbol{o}$ output

Example of a 2 hidden layer neural network (or 4 layer network, counting also input and output).

Forward Propagation

Def.: Forward propagation is the process of computing the output of the network given its input.

Forward Propagation



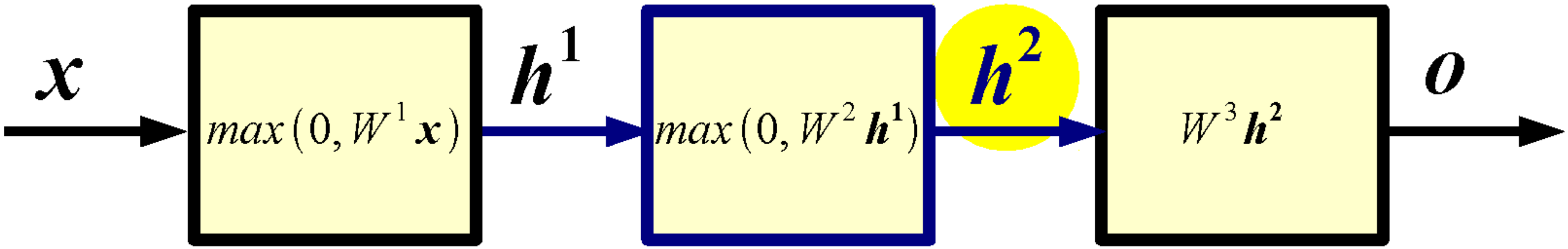
$\boldsymbol{x} \in R^D \quad W^1 \in R^{N_1 \times D} \quad \boldsymbol{b}^1 \in R^{N_1} \quad \boldsymbol{h}^1 \in R^{N_1}$

$$\boldsymbol{h}^1 = max(0, W^1 \boldsymbol{x} + \boldsymbol{b}^1)$$

W^1 1-st layer weight matrix or weights
$\boldsymbol{b}^1$ 1-st layer biases

The non-linearity $u = max(0, v)$ is called **ReLU** in the DL literature. Each output hidden unit takes as input all the units at the previous layer: each such layer is called "**fully connected**".

Forward Propagation



$h^1 \in R^{N_1}$ $W^2 \in R^{N_2 \times N_1}$ $b^2 \in R^{N_2}$ $h^2 \in R^{N_2}$

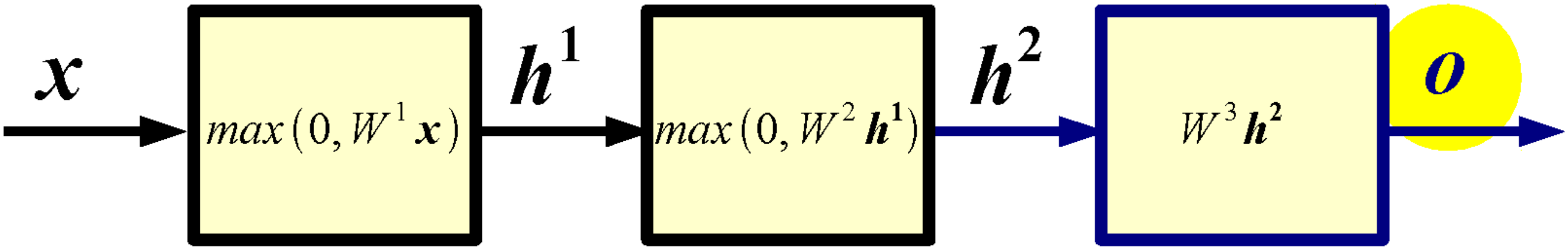
$$h^2 = max(0, W^2 h^1 + b^2)$$

W^2 2-nd layer weight matrix or weights

b^2 2-nd layer biases

Ranzato

Forward Propagation



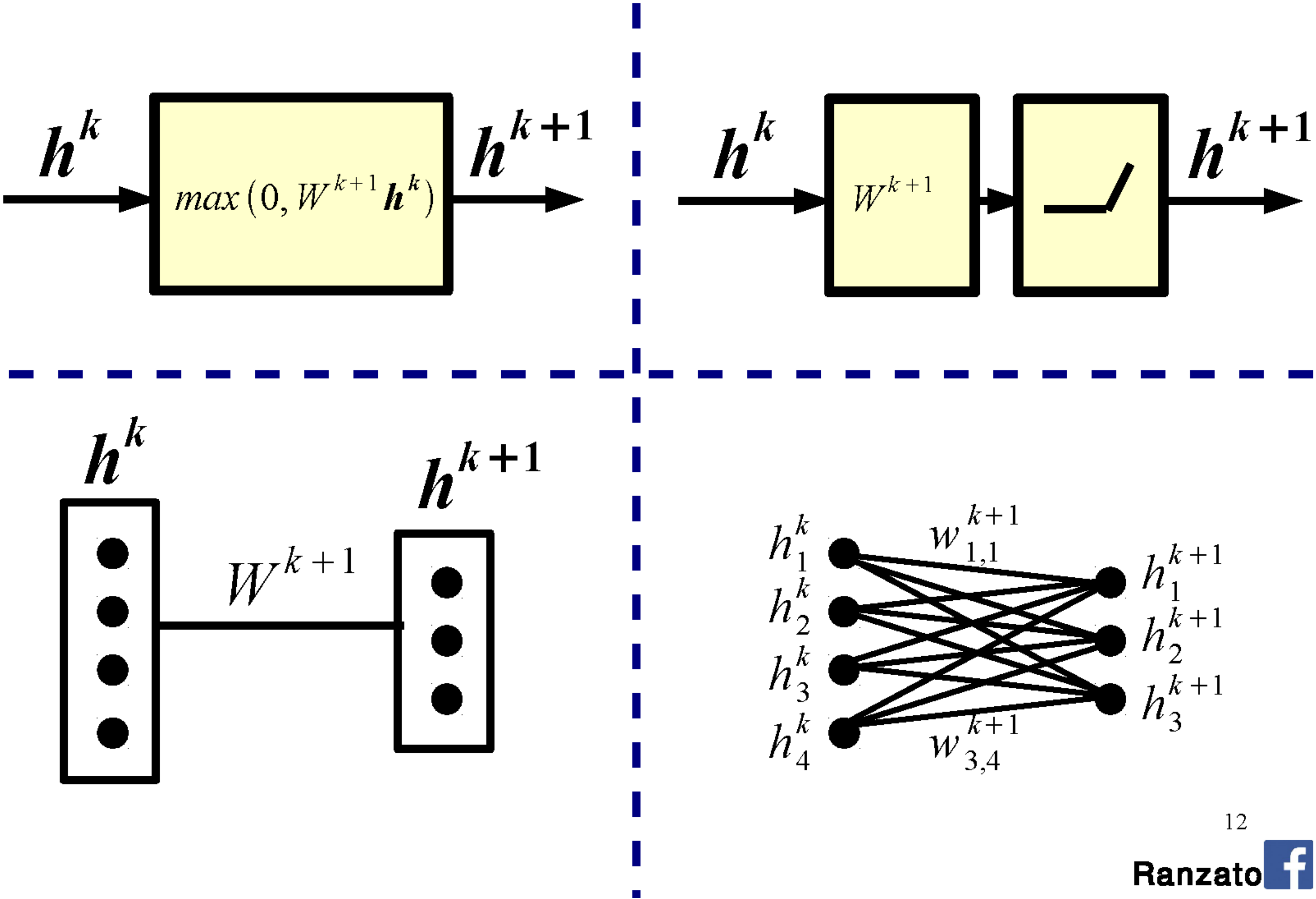
$\boldsymbol{h}^2 \in R^{N_2}$ $\quad W^3 \in R^{N_3 \times N_2}$ $\quad \boldsymbol{b}^3 \in R^{N_3}$ $\quad \boldsymbol{o} \in R^{N_3}$

$$\boldsymbol{o} = max(0, W^3 \boldsymbol{h}^2 + \boldsymbol{b}^3)$$

W^3 3-rd layer weight matrix or weights

$\boldsymbol{b}^3$ 3-rd layer biases

Alternative Graphical Representation



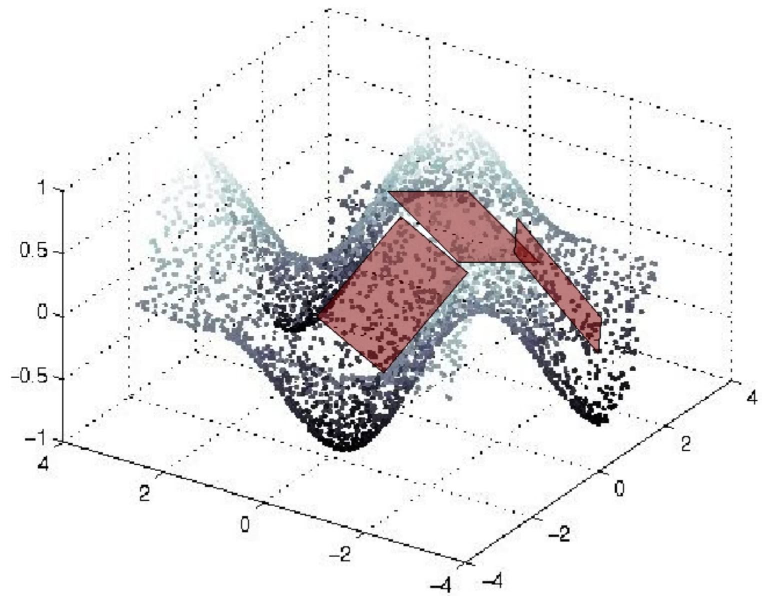
Interpretation

Question: Why can't the mapping between layers be linear?

Answer: Because composition of linear functions is a linear function. Neural network would reduce to (1 layer) logistic regression.

Question: What do ReLU layers accomplish?

Answer: Piece-wise linear tiling: mapping is locally linear.

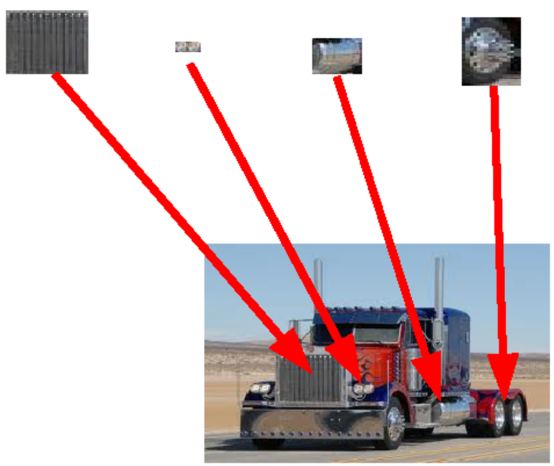


Interpretation

Question: Why do we need many layers?

Answer: When input has hierarchical structure, the use of a hierarchical architecture is potentially more efficient because intermediate computations can be re-used. DL architectures are efficient also because they use **distributed representations** which are shared across classes.

[0 0 **1** 0 0 0 0 **1** 0 0 **1** **1** 0 0 **1** 0 …] truck feature



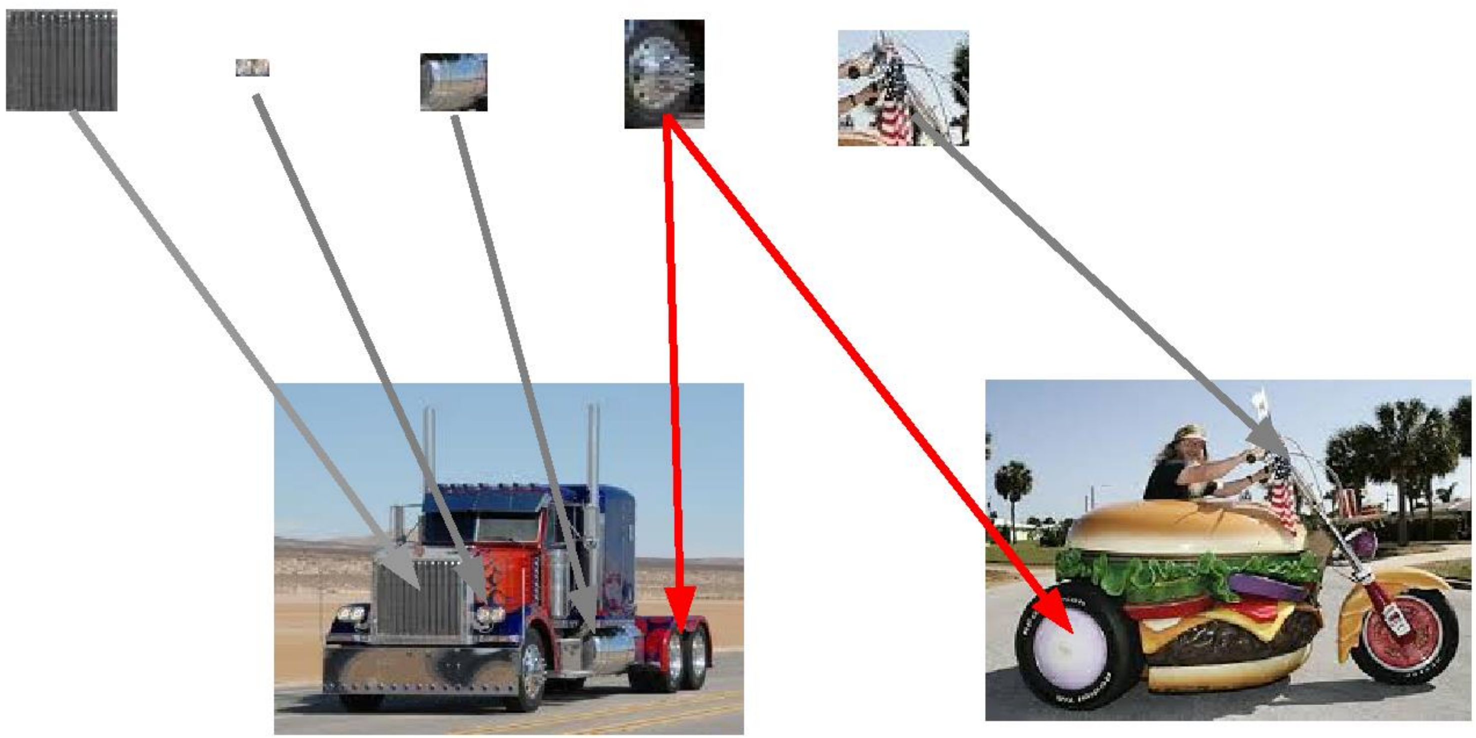
Exponentially more efficient than a 1-of-N representation (a la k-means)

Ranzato

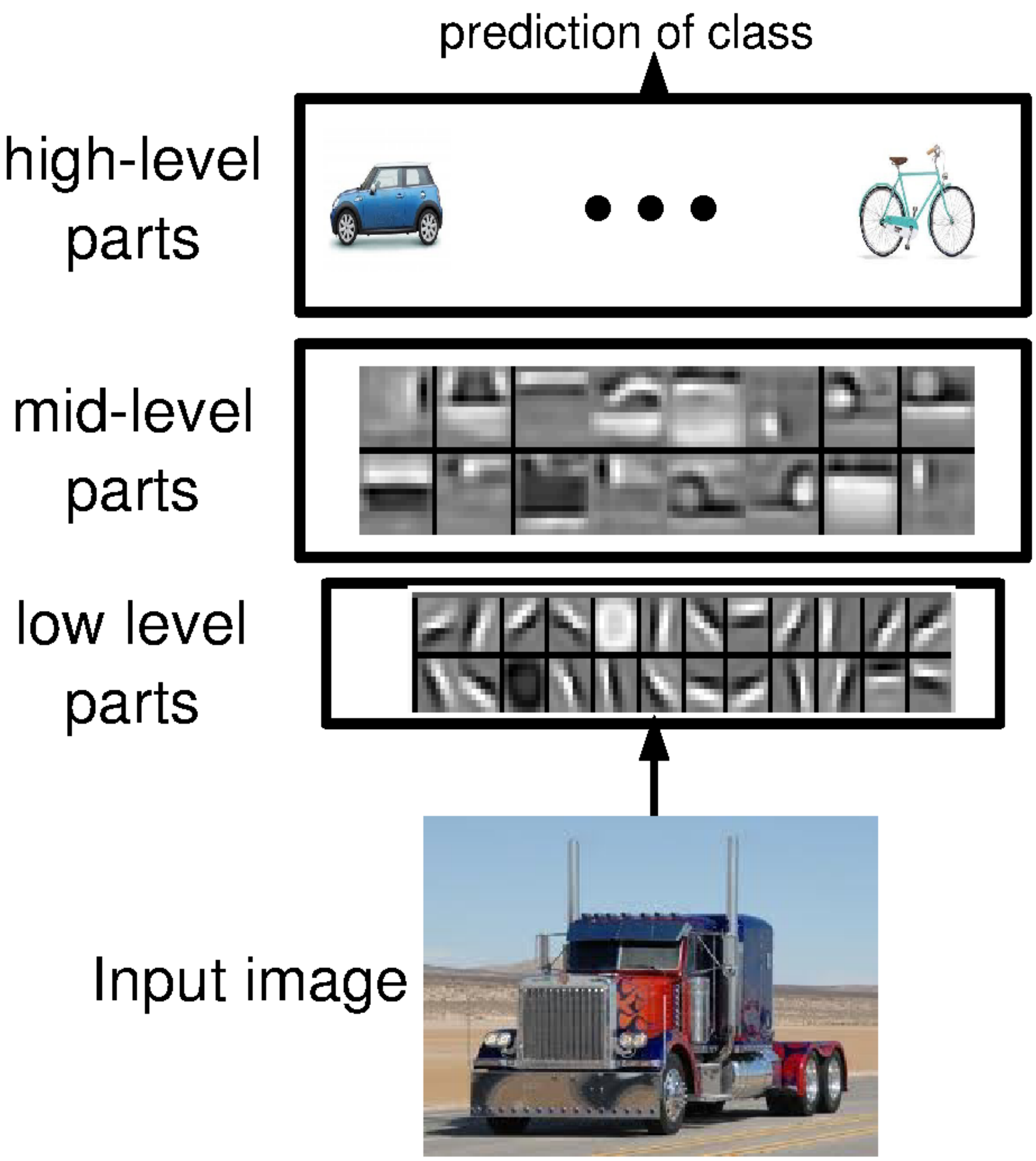
Interpretation

[1 1 0 0 0 1 0 **1** 0 0 0 0 1 1 0 1…] motorbike

[0 0 1 0 0 0 0 **1** 0 0 1 1 0 0 1 0 …] truck



Interpretation



- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009

Interpretation

Question: What does a hidden unit do?

Answer: It can be thought of as a classifier or feature detector.

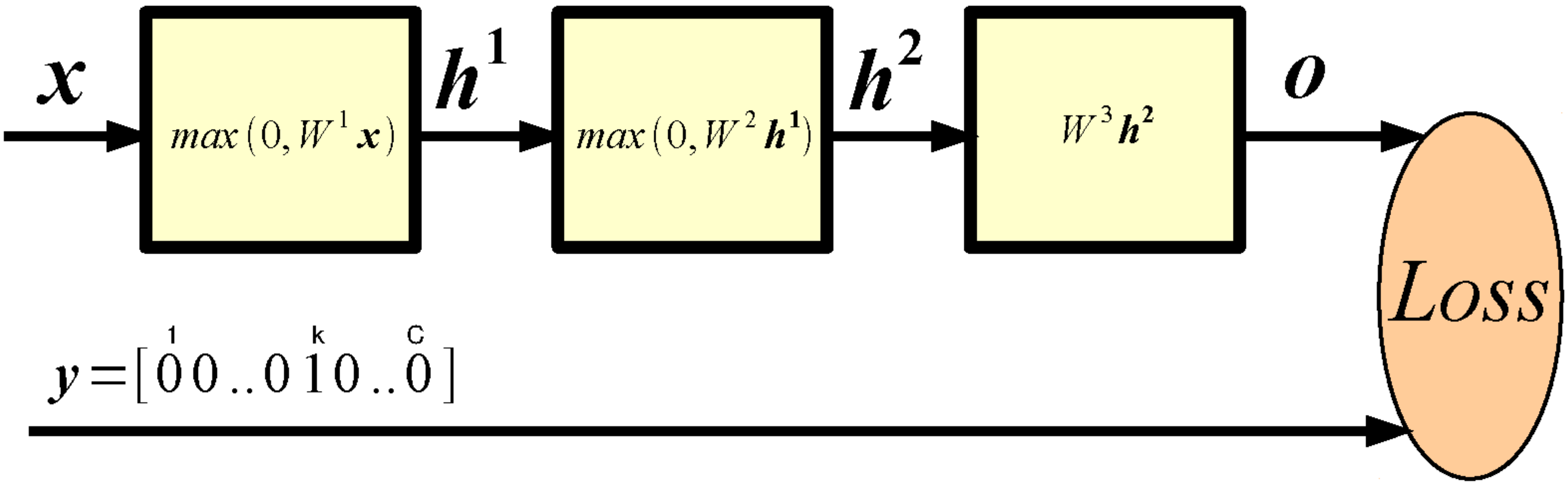
Question: How many layers? How many hidden units?

Answer: Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

Question: How do I set the weight matrices?

Answer: Weight matrices and biases are learned.
First, we need to define a measure of quality of the current mapping.
Then, we need to define a procedure to adjust the parameters.

How Good is 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):

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

Training

Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

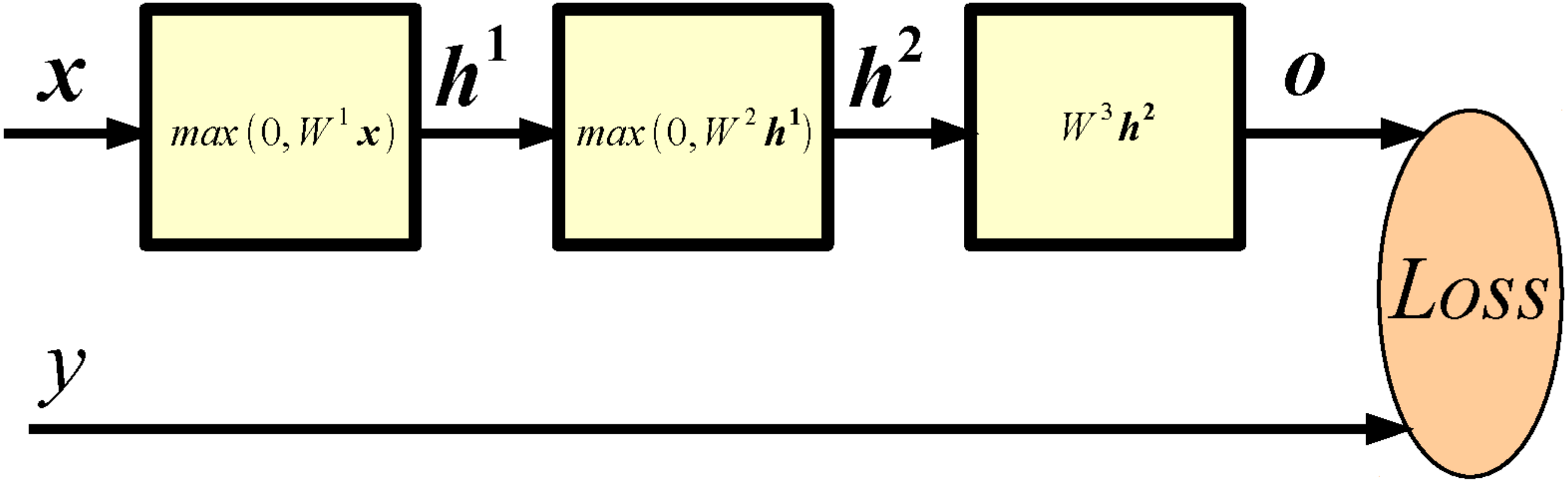
$$\boldsymbol{\theta}^* = arg\, min_{\boldsymbol{\theta}} \sum_{n=1}^{P} L(\boldsymbol{x}^n, y^n; \boldsymbol{\theta})$$

Question: How to minimize a complicated function of the parameters?

Answer: Chain rule, a.k.a. **Backpropagation**! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.

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

Key Idea: Wiggle To Decrease Loss



Let's say we want to decrease the loss by adjusting $W^1_{i,j}$.
We could consider a very small $\epsilon = 1\text{e-}6$ and compute:

$$L(\boldsymbol{x}, y; \boldsymbol{\theta})$$

$$L(\boldsymbol{x}, y; \boldsymbol{\theta} \setminus W^1_{i,j}, W^1_{i,j} + \epsilon)$$

Then, update:

$$W^1_{i,j} \leftarrow W^1_{i,j} + \epsilon\, sgn(L(\boldsymbol{x}, y; \boldsymbol{\theta}) - L(\boldsymbol{x}, y; \boldsymbol{\theta} \setminus W^1_{i,j}, W^1_{i,j} + \epsilon))$$

Derivative w.r.t. Input of Softmax

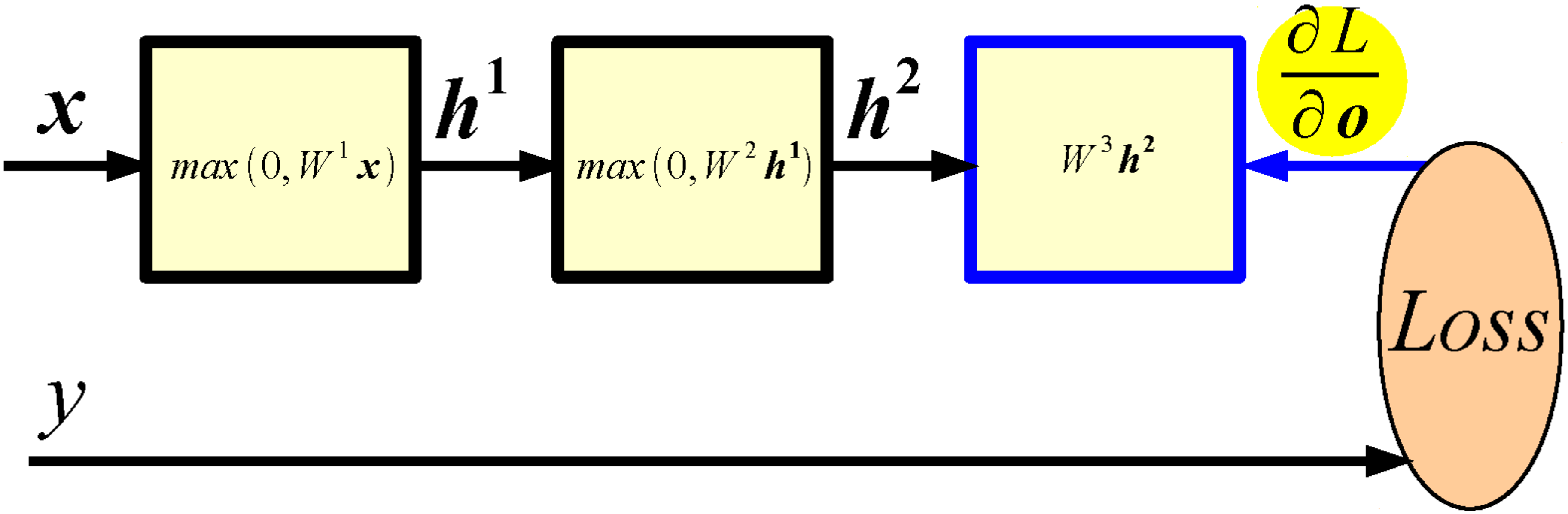
$$p(c_k=1|\boldsymbol{x})=\frac{e^{o_k}}{\sum_j e^{o_j}}$$

$$L(\boldsymbol{x},y;\boldsymbol{\theta})=-\sum_j y_j \log p(c_j|\boldsymbol{x}) \qquad \boldsymbol{y}=[\overset{1}{0}\;0\;..\;0\;\overset{k}{1}\;0\;..\;\overset{C}{0}]$$

By substituting the fist formula in the second, and taking the derivative w.r.t. $\boldsymbol{o}$ we get:

$$\frac{\partial L}{\partial o}=p(c|\boldsymbol{x})-\boldsymbol{y}$$

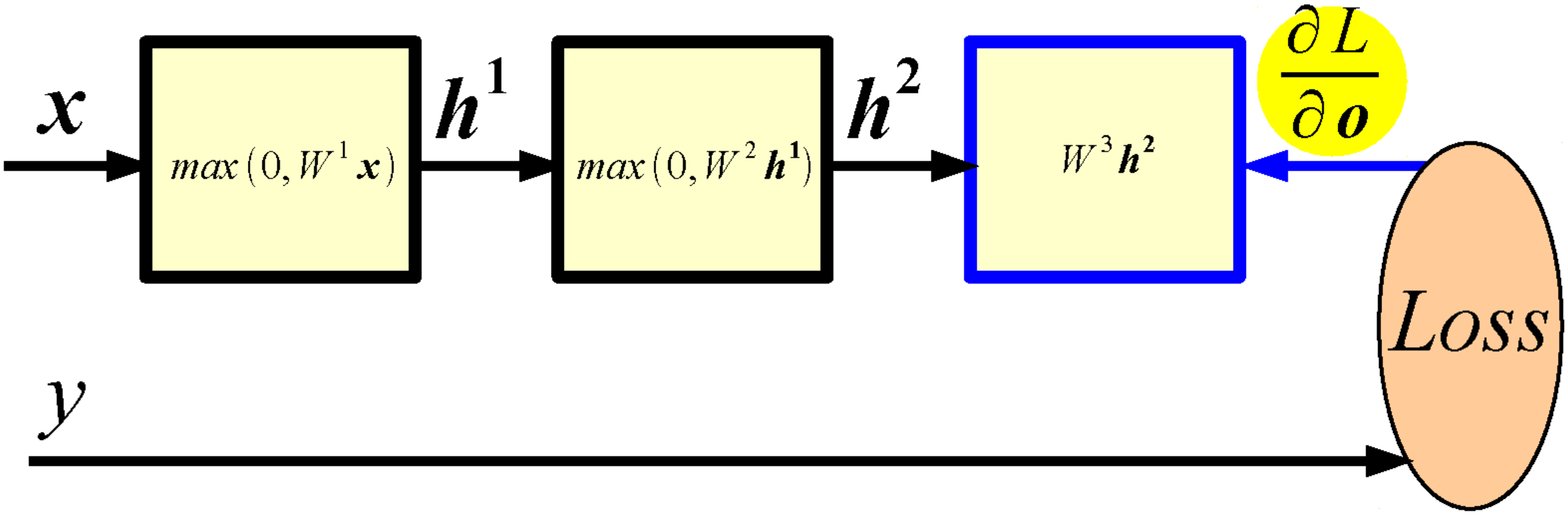
Backward Propagation



Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

Backward Propagation

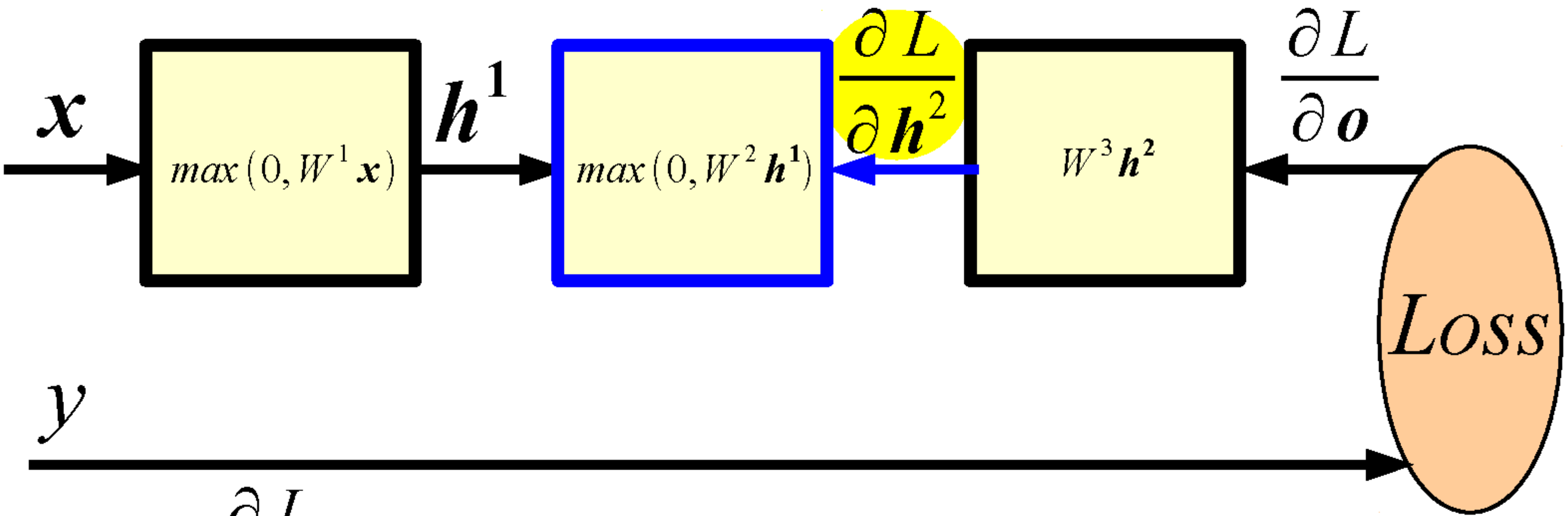


Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\boldsymbol{x}) - \boldsymbol{y}) \; \boldsymbol{h}^{2\,T} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = W^{3\,T} (p(c|\boldsymbol{x}) - \boldsymbol{y})$$

Backward Propagation

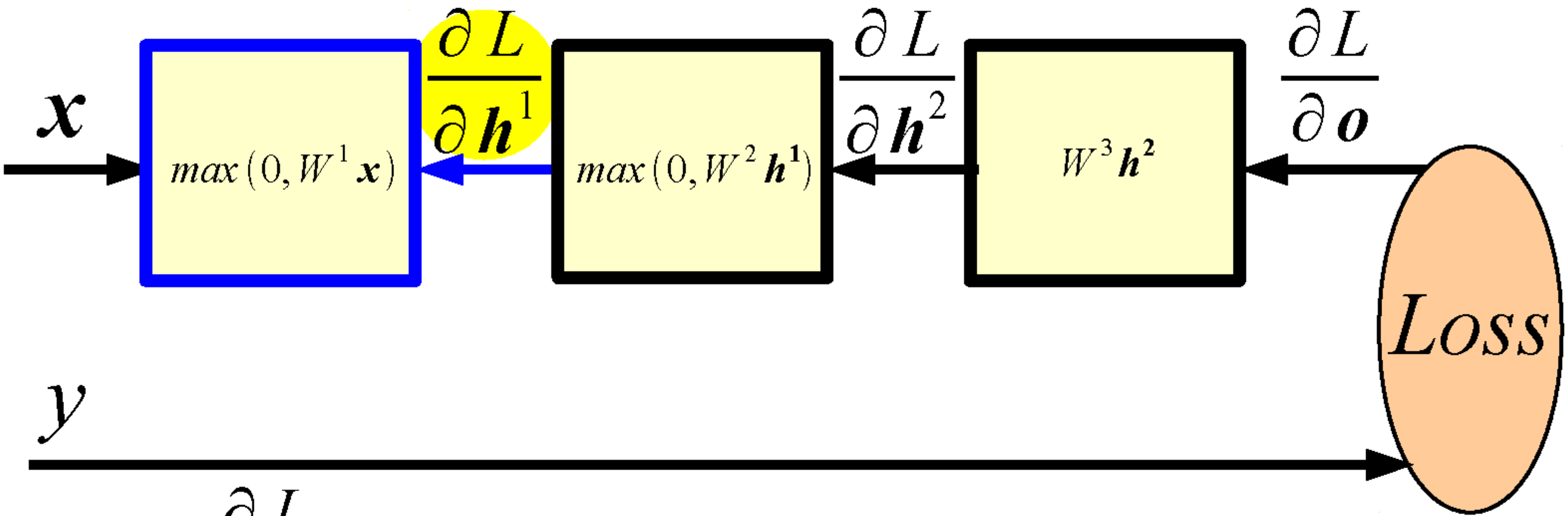


Given $\frac{\partial L}{\partial \boldsymbol{h}^2}$ we can compute now:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial \boldsymbol{h}^2} \frac{\partial \boldsymbol{h}^2}{\partial W^2} \qquad \frac{\partial L}{\partial \boldsymbol{h}^1} = \frac{\partial L}{\partial \boldsymbol{h}^2} \frac{\partial \boldsymbol{h}^2}{\partial \boldsymbol{h}^1}$$

Ranzato

Backward Propagation



Given $\frac{\partial L}{\partial \boldsymbol{h}^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial \boldsymbol{h}^1} \frac{\partial \boldsymbol{h}^1}{\partial W^1}$$

Ranzato

Backward Propagation

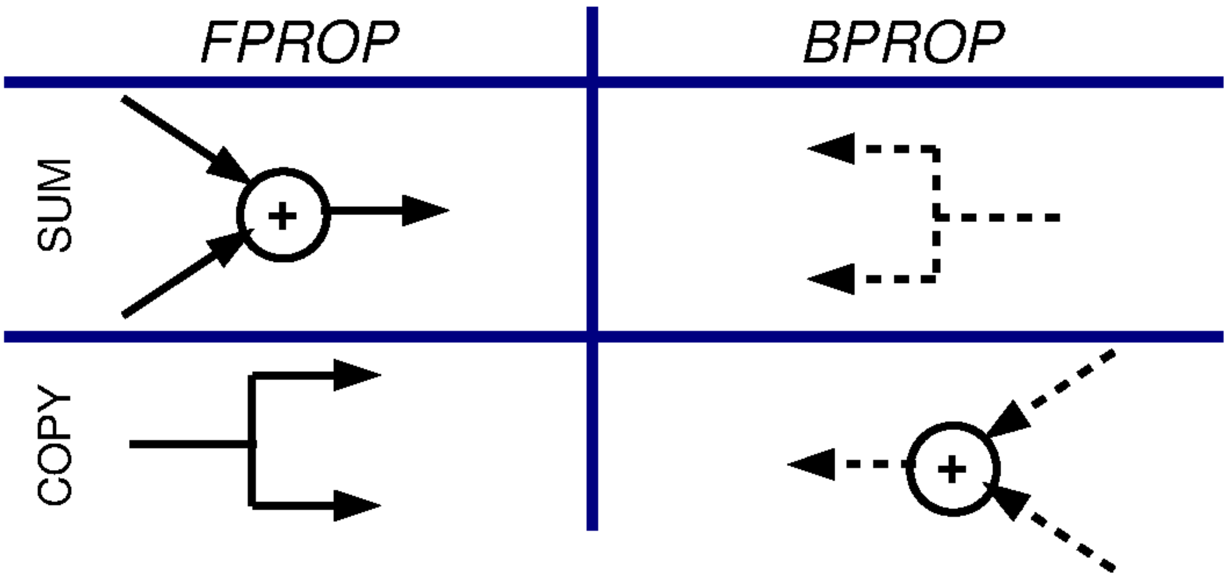
Question: Does BPROP work with ReLU layers only?

Answer: Nope, any a.e. differentiable transformation works.

Question: What's the computational cost of BPROP?

Answer: About twice FPROP (need to compute gradients w.r.t. input and parameters at every layer).

Note: FPROP and BPROP are dual of each other. E.g.,:



Optimization

Stochastic Gradient Descent (on mini-batches):

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial L}{\partial \boldsymbol{\theta}}, \eta \in (0, 1)$$

Stochastic Gradient Descent with Momentum:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{\Delta}$$

$$\boldsymbol{\Delta} \leftarrow 0.9 \boldsymbol{\Delta} + \frac{\partial L}{\partial \boldsymbol{\theta}}$$

Note: there are many other variants...

Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips