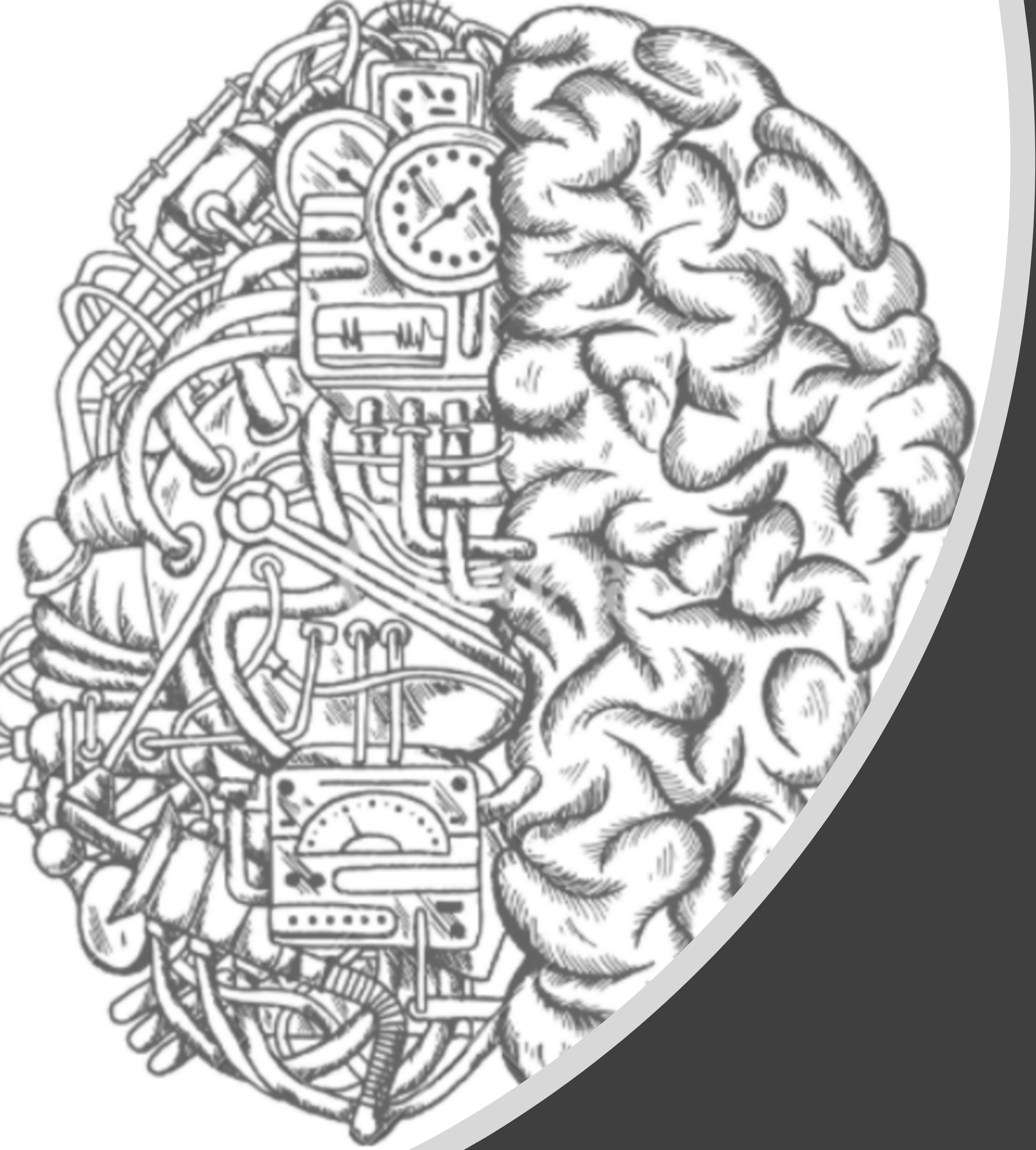


CS 559: Machine Learning Fundamentals and Applications

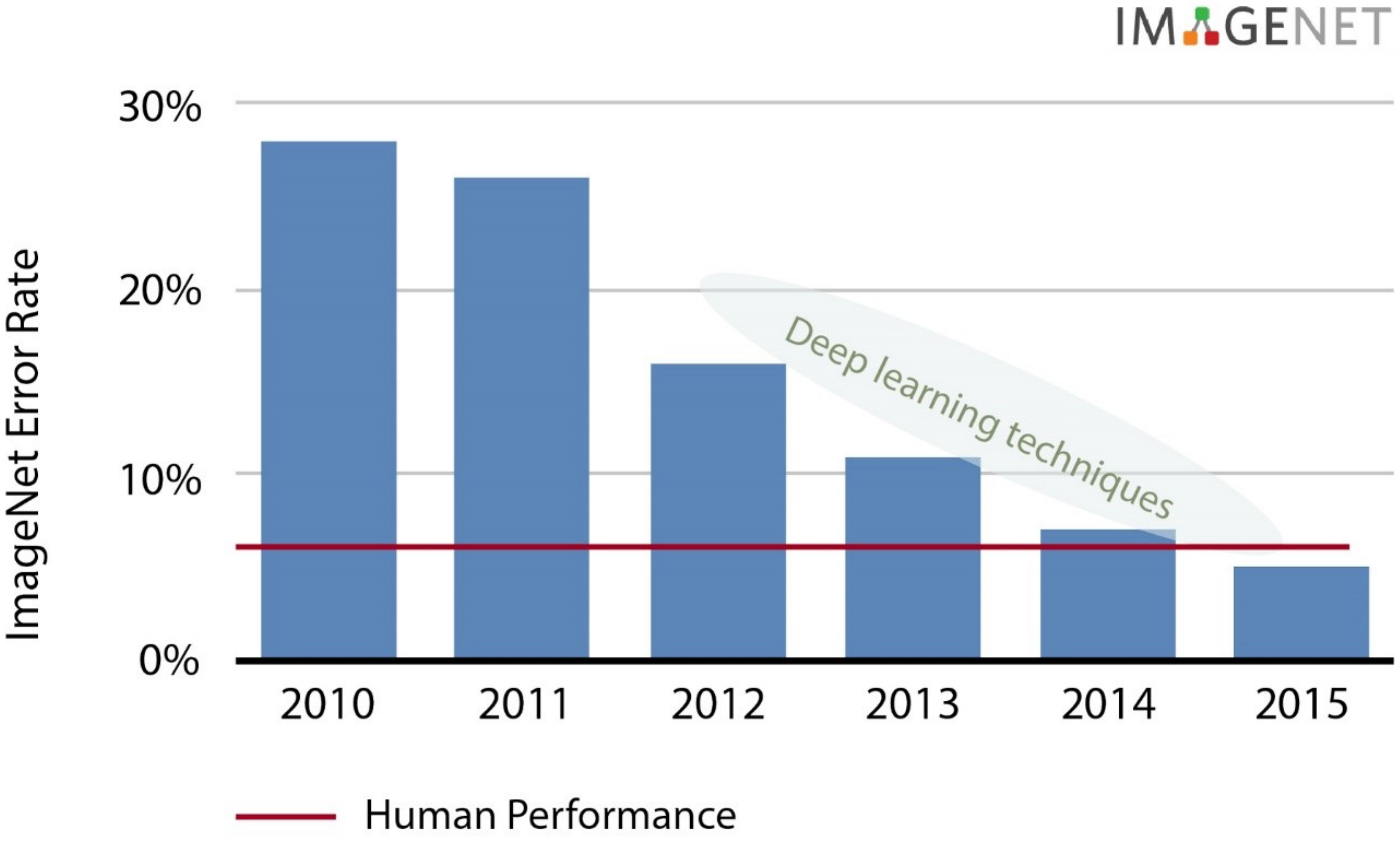
Lecture 11

Lecturer: Xinchao Wang

xinchao.wang@stevens.edu

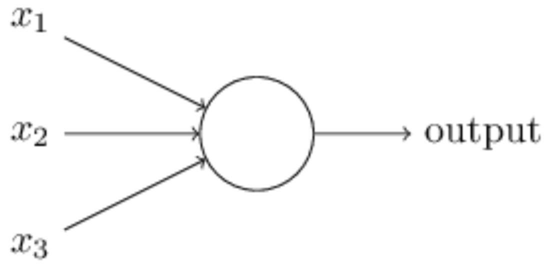
Teaching Assistant: Yiding Yang

yyang99@stevens.edu


IMAGENET
30%
20%
10%
0%
ImageNet Error Rate
Deep learning techniques
2010
2011
2012
2013
2014
2015
Human Performance

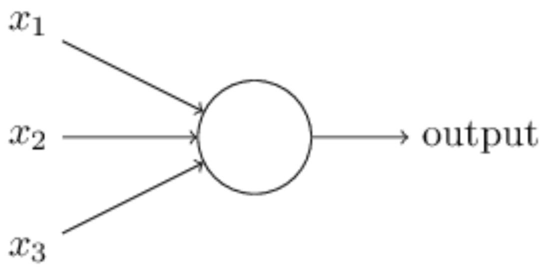
Perceptrons

- Perceptrons
  - 1950s ~ 1960s, Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts
- Standard model of artificial neurons



Binary Perceptrons

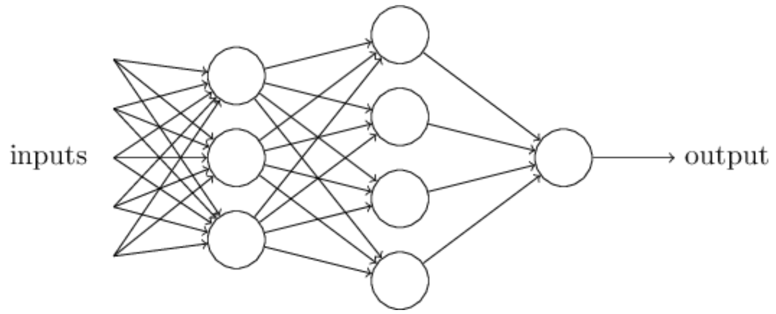
- Inputs
  - Multiple binary inputs
- Parameters
  - Thresholds & weights
- Outputs
  - Thresholded weighted linear combination



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Layered Perceptrons

- Layered, complex model
  - 1st layer, 2nd layer of perceptrons
- Perceptron rule
  - Weights, thresholds



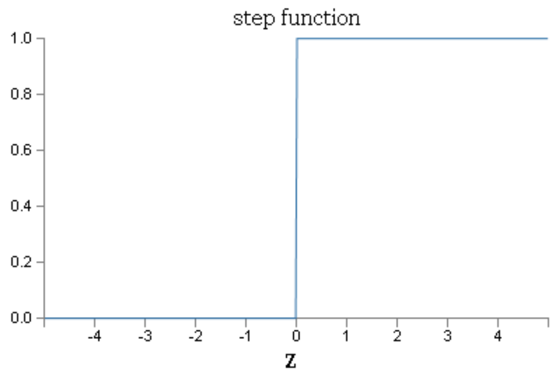
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

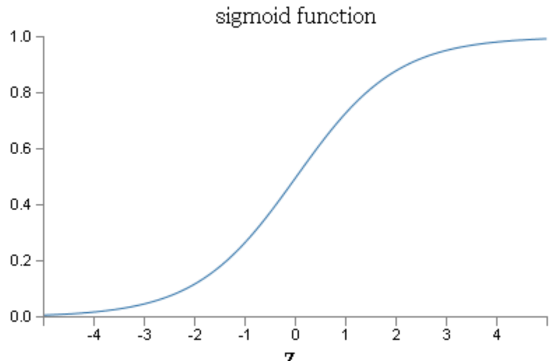
Output Functions

- Sigmoid neurons
- Output $\sigma(w \cdot x + b)$, $\quad \sigma(z) \equiv \frac{1}{1+e^{-z}}$

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

- Sigmoid vs conventional thresholds





Neural Nets for Computer Vision

Based on Tutorials at CVPR 2012 and 2014 by

Marc'Aurelio Ranzato

Key Ideas of Neural Nets

IDEA # 1

Learn features from data

IDEA # 2

Use differentiable functions that produce features efficiently

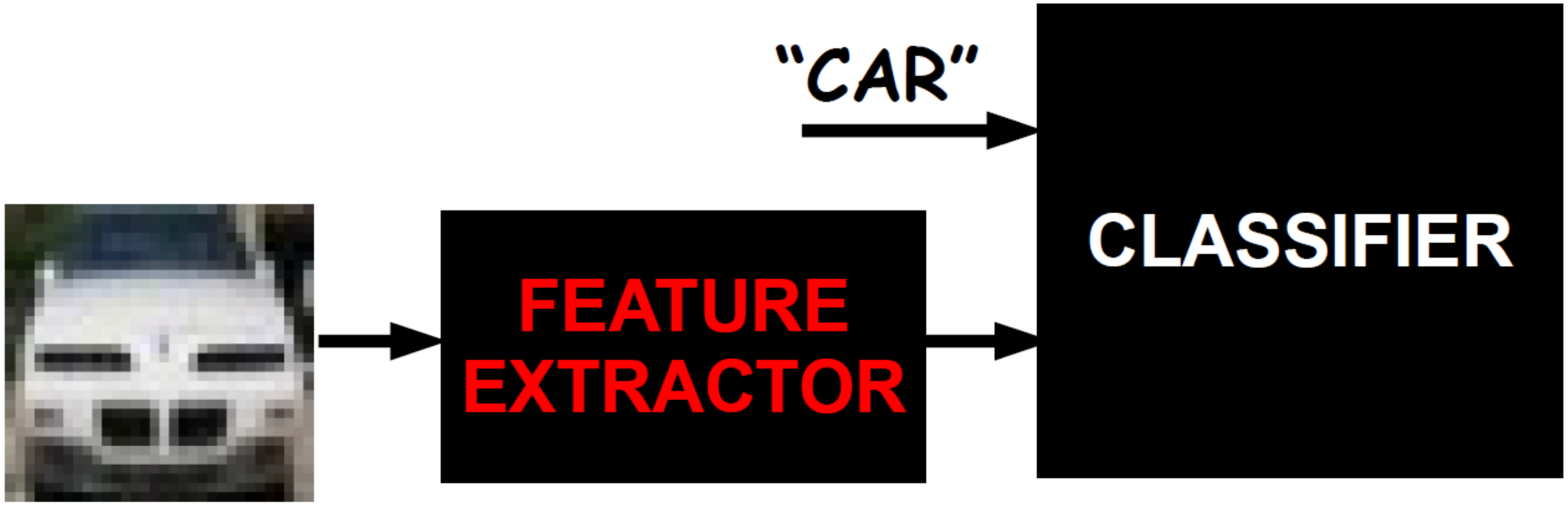
IDEA # 3

End-to-end learning:
no distinction between feature extractor and classifier

IDEA # 4

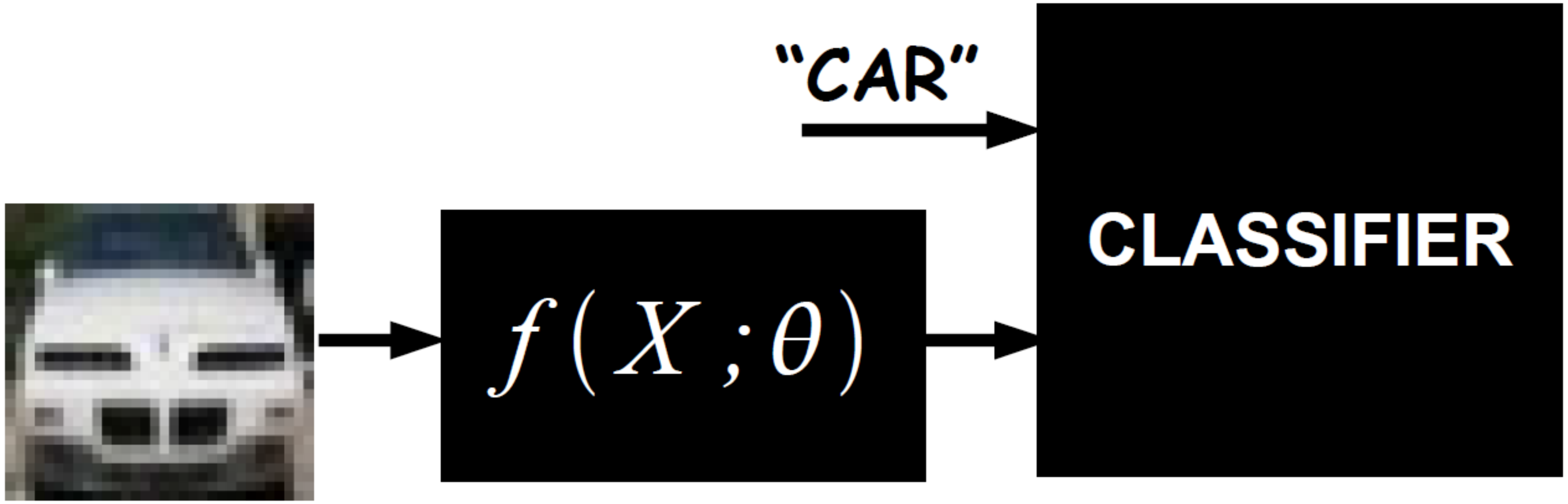
“Deep” architectures:
cascade of simpler non-linear modules

Building an Object Recognition System



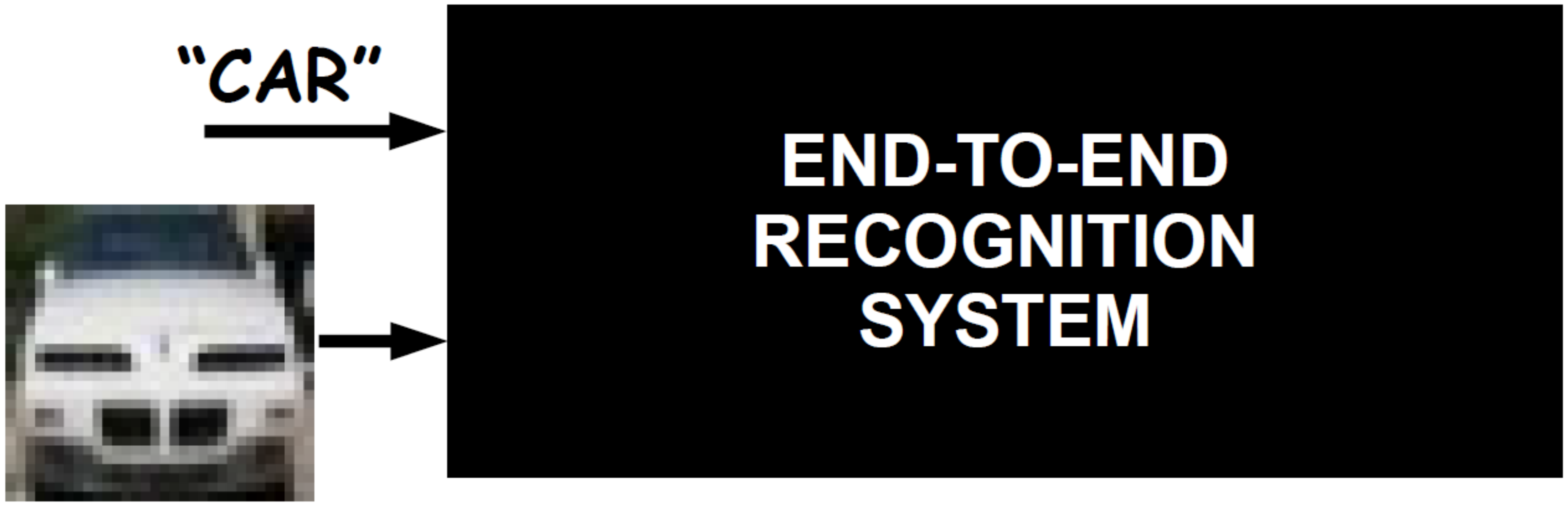
IDEA: Use data to optimize features for the given task

Building an Object Recognition System



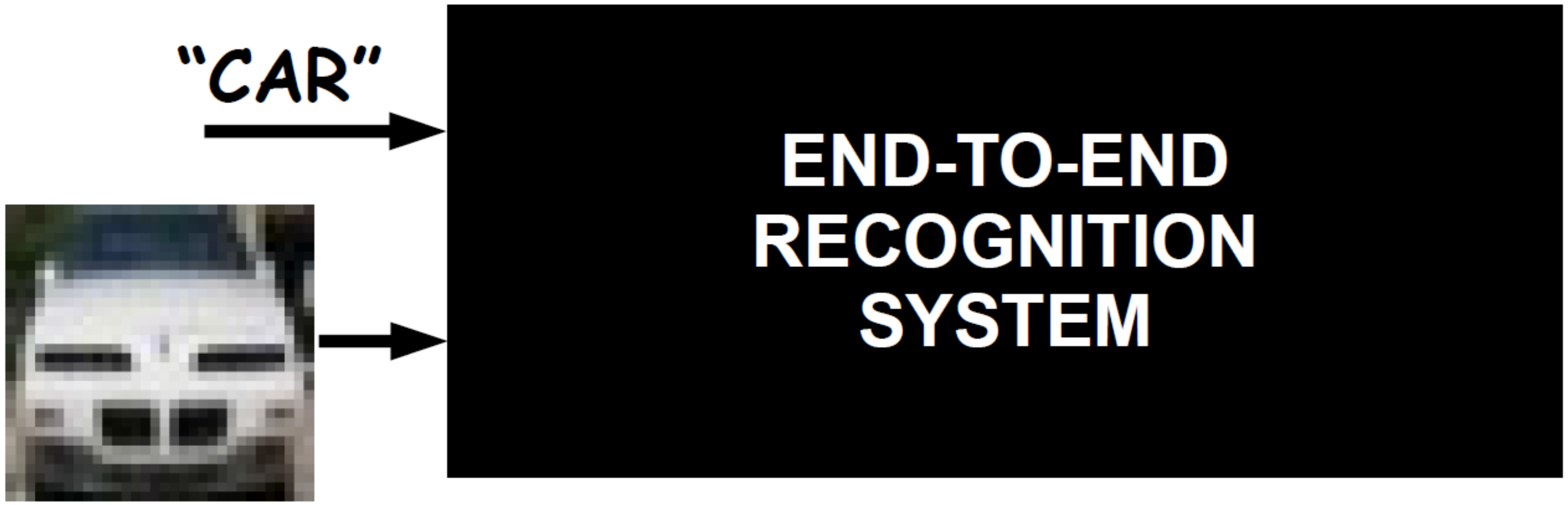
What we want: Use parameterized function such that
a) features are computed efficiently
b) features can be trained efficiently

Building an Object Recognition System



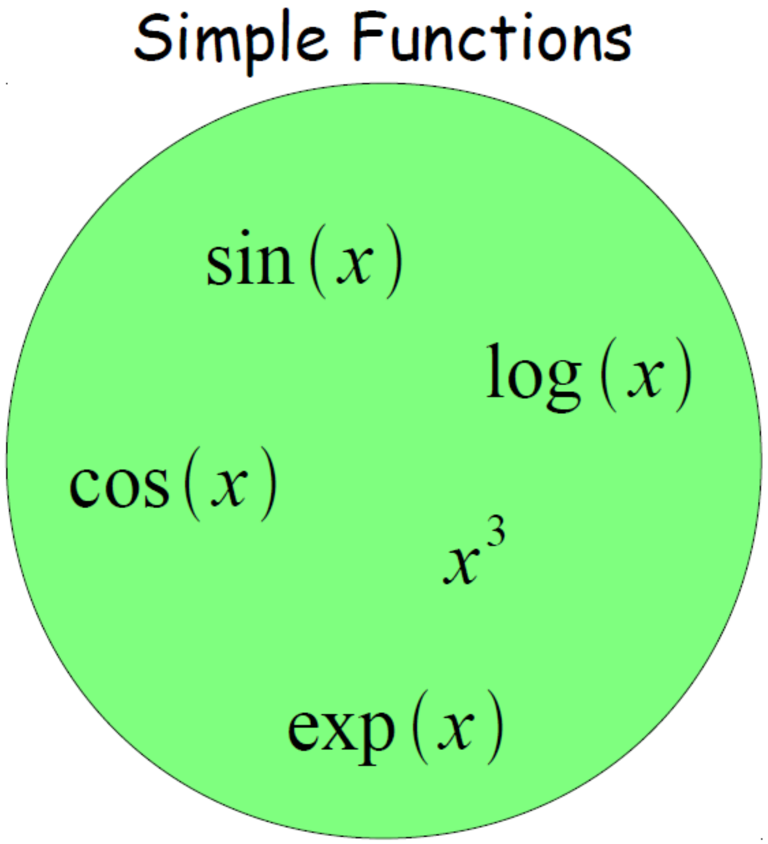
- Everything becomes adaptive
- No distinction between feature extractor and classifier
- Big non-linear system trained from raw pixels to labels

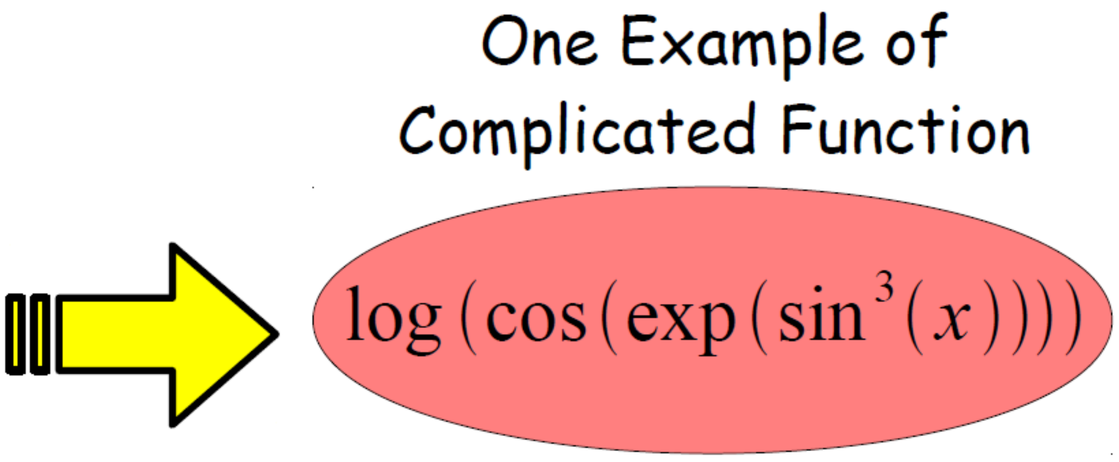
Building an Object Recognition System



Q: How can we build such a highly non-linear system?
A: By combining simple building blocks we can make more and more complex systems

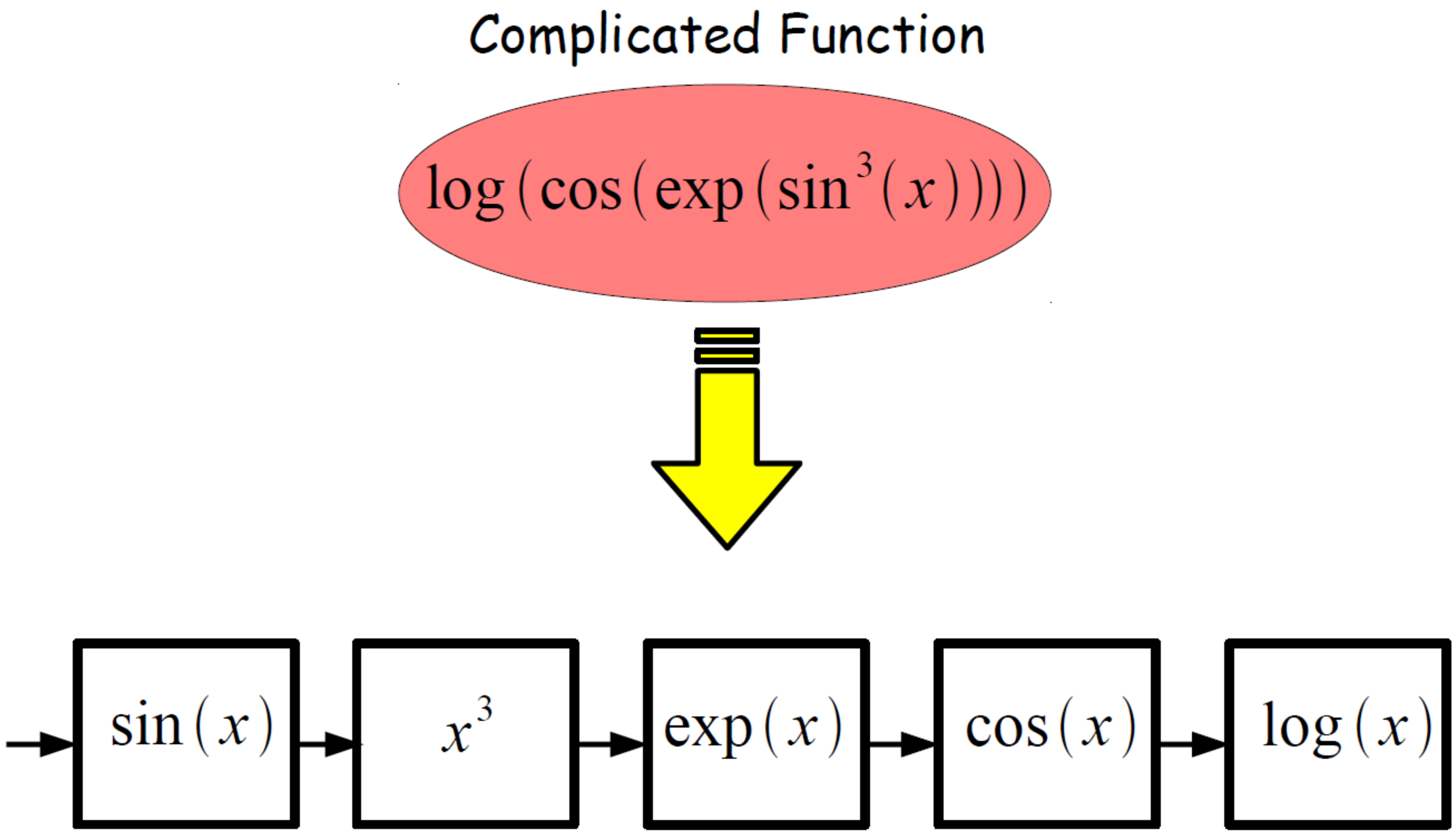
Building a Complicated Function





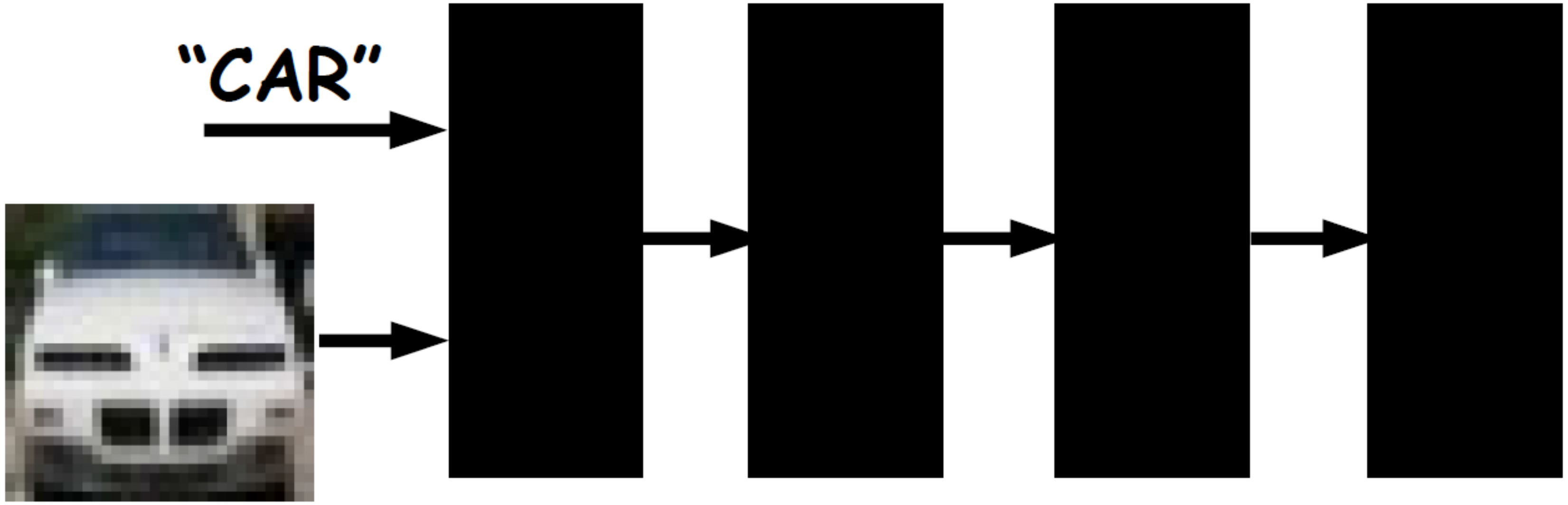
- Function composition is at the core of deep learning methods
- Each “simple function” will have parameters subject to training

Implementing a Complicated Function



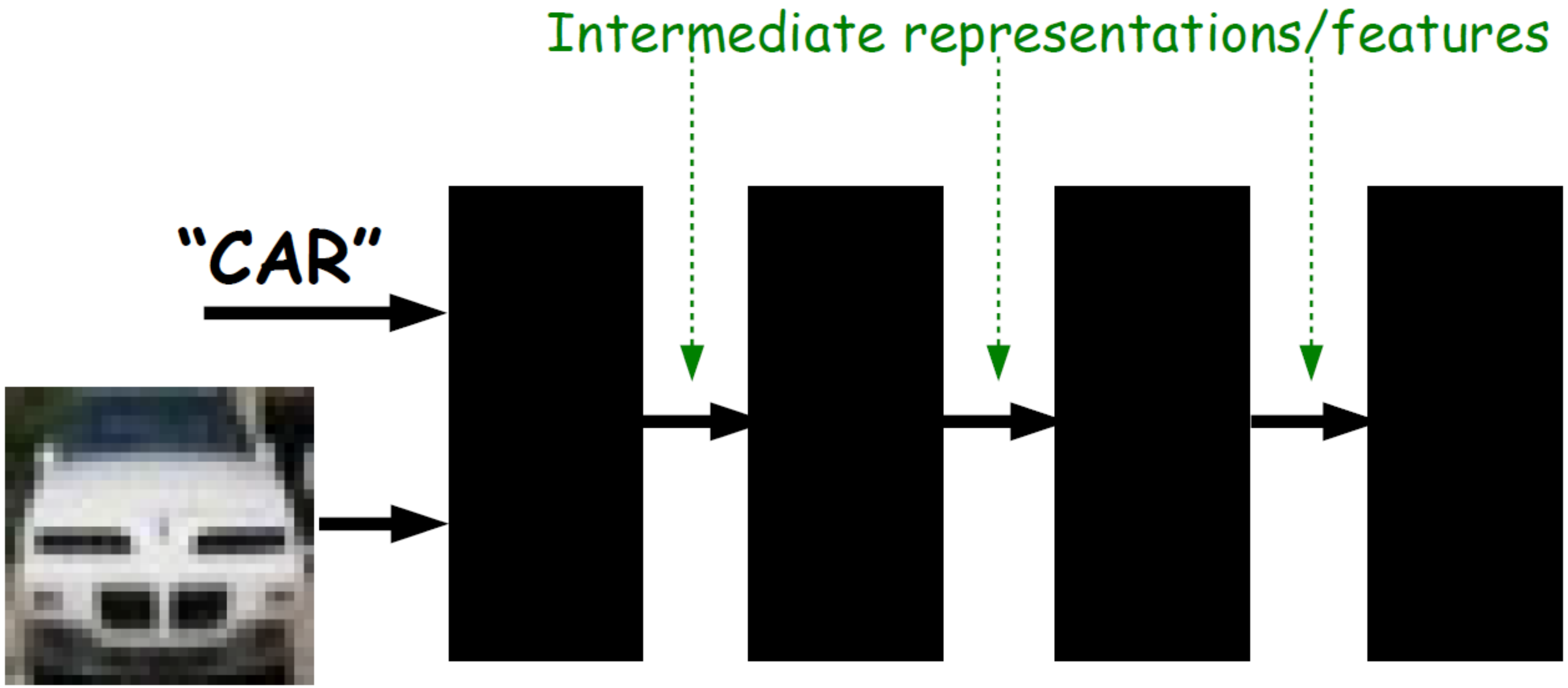
Intuition Behind Deep Neural Nets

Intuition Behind Deep Neural Nets



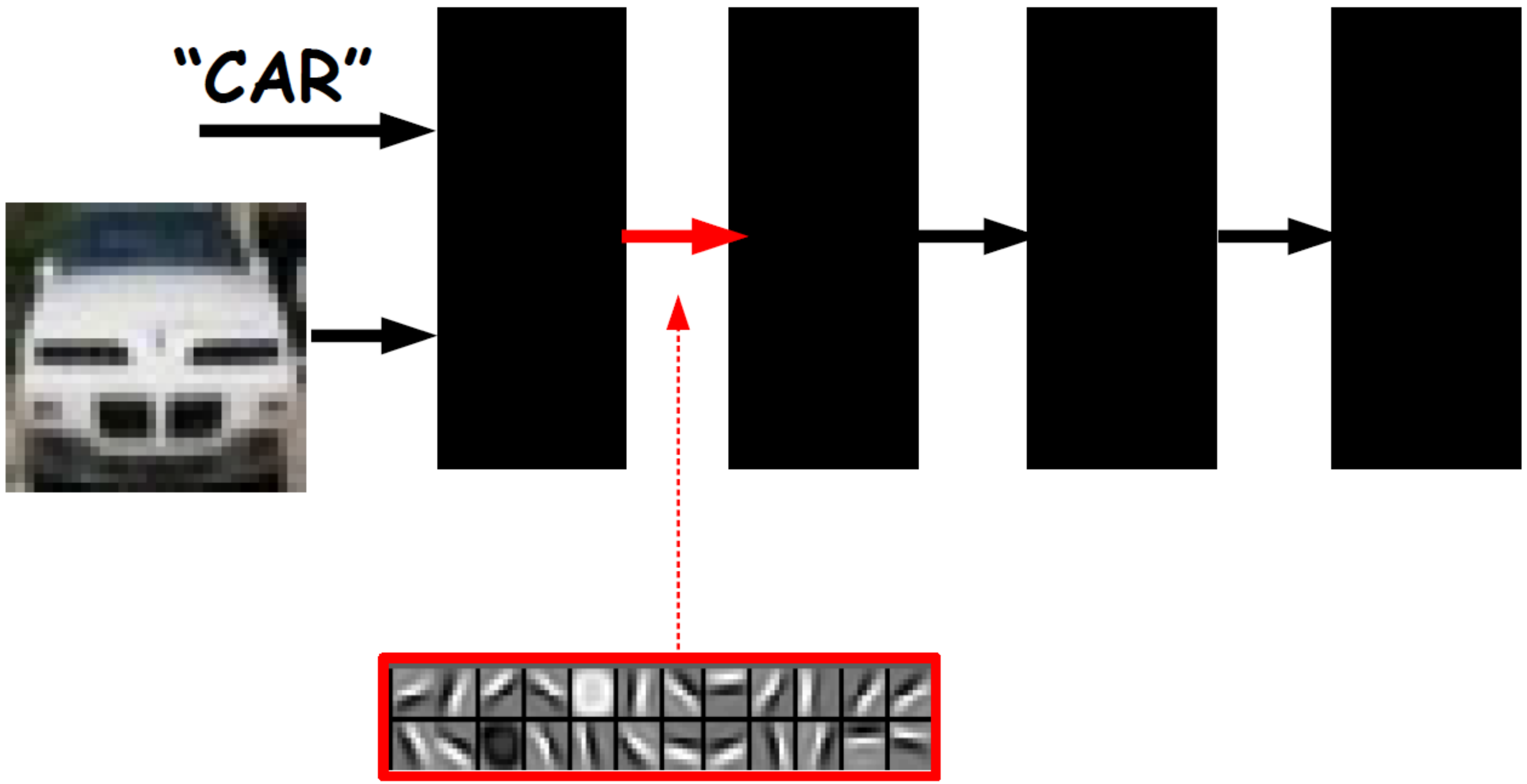
Each black box can have trainable parameters. Their composition makes a highly non-linear system.

Intuition Behind Deep Neural Nets

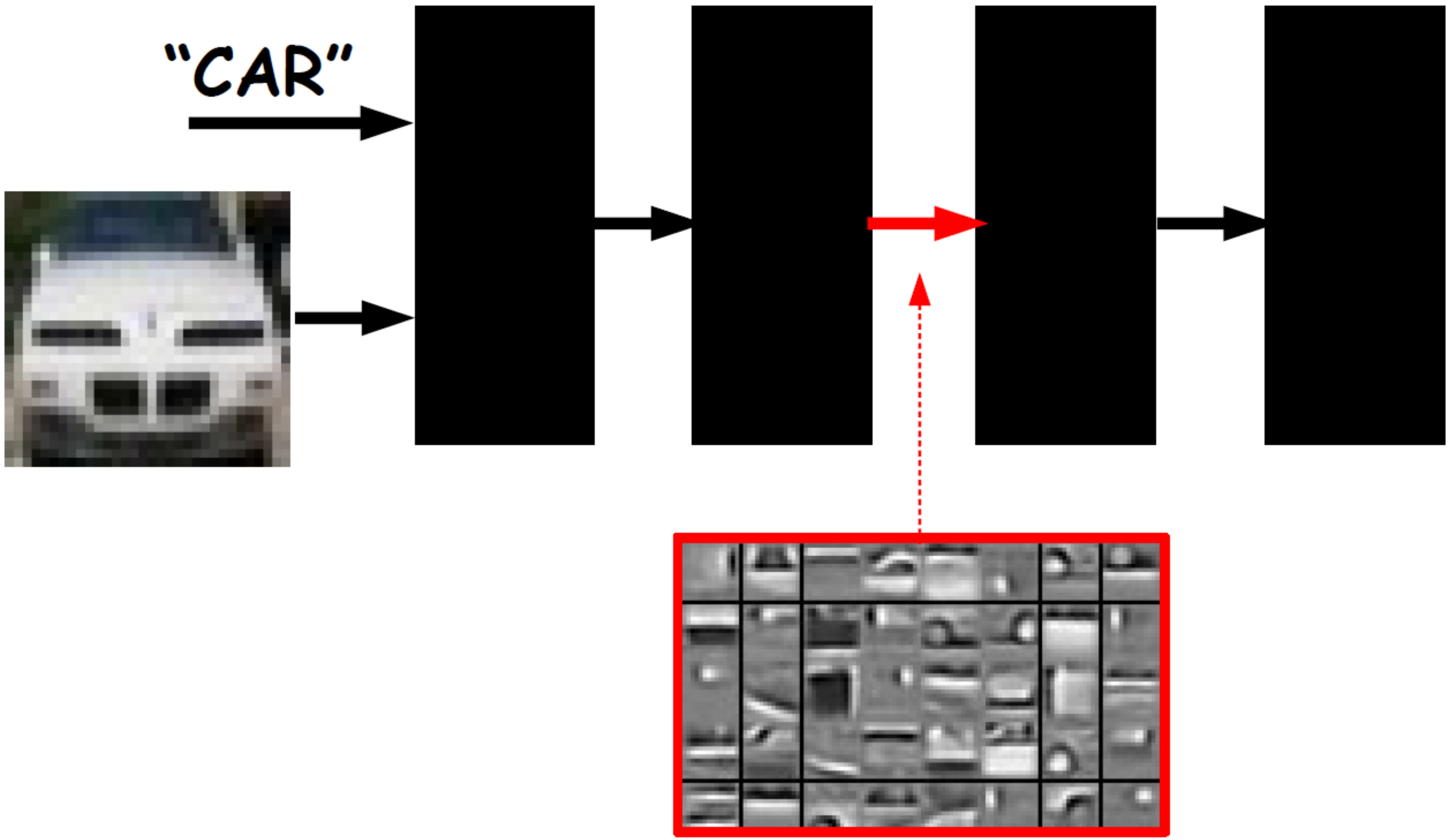


System produces hierarchy of features

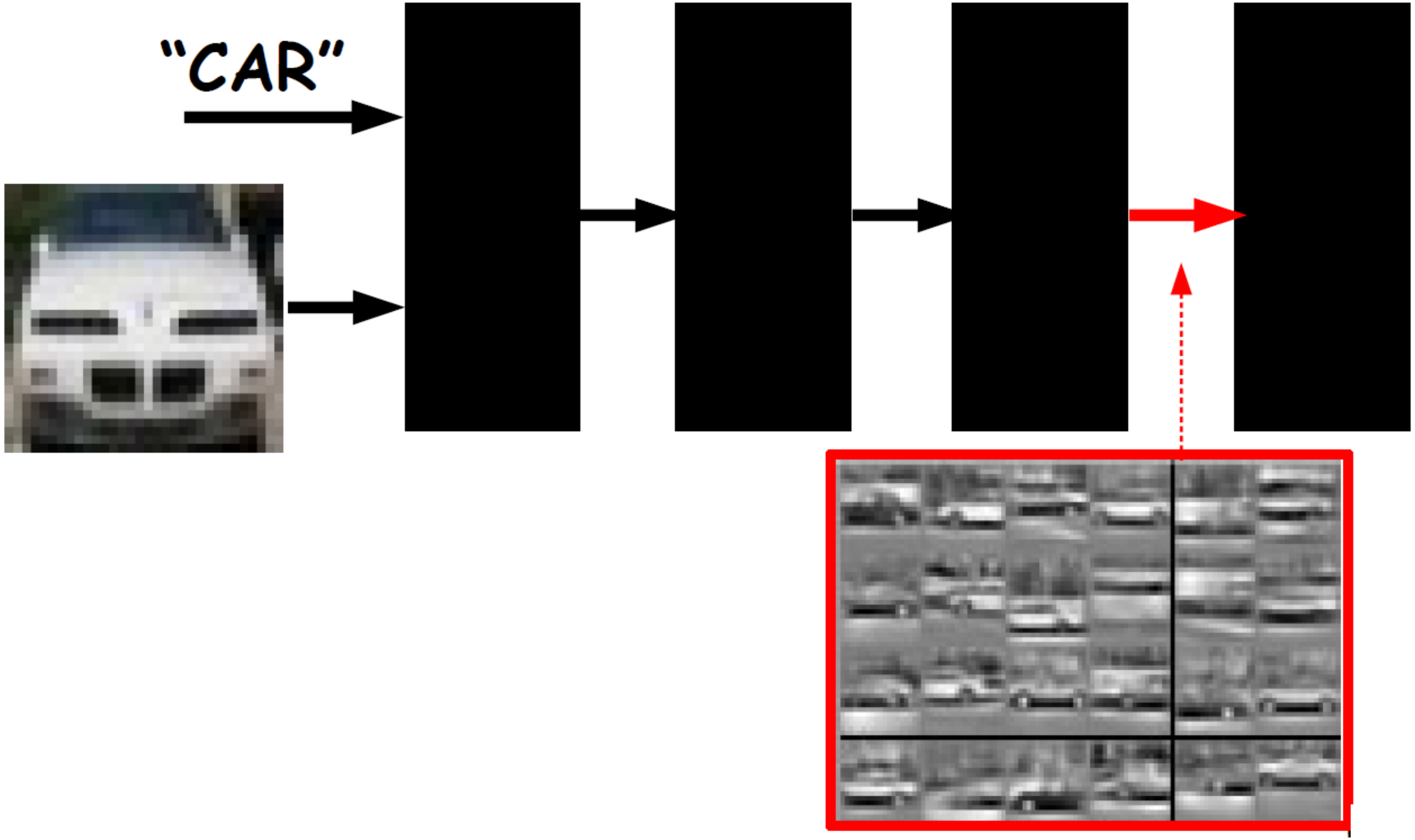
Intuition Behind Deep Neural Nets



Intuition Behind Deep Neural Nets



Intuition Behind Deep Neural Nets



Key Questions

- What is the input-output mapping?
- How are parameters trained?
- How computational expensive is it?
- How well does it work?

Supervised Deep Learning

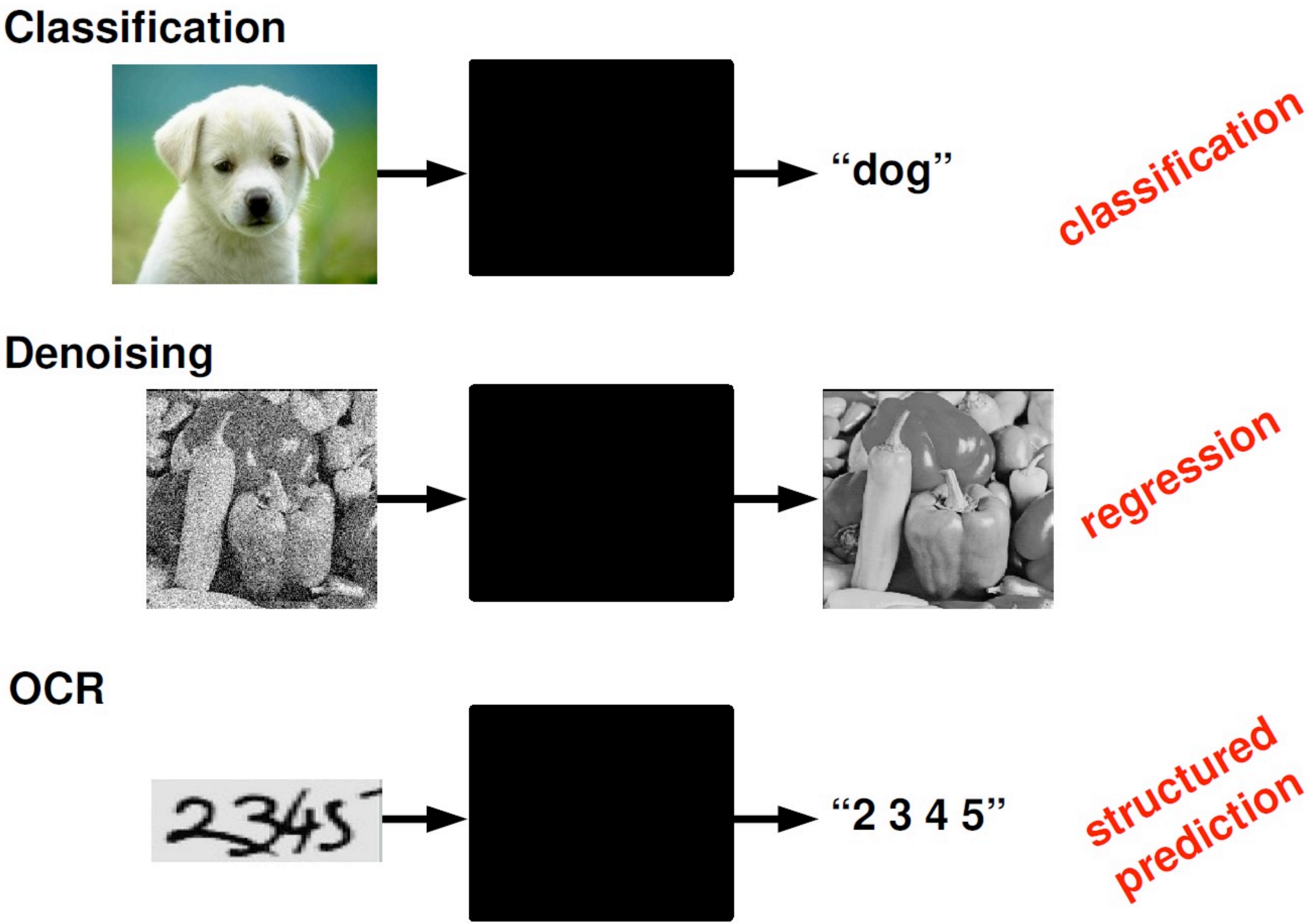
Marc'Aurelio Ranzato

Supervised Learning

$\{(x_i, y_i), i=1... P\}$ training set
x_i i-th input training example
y_i i-th target label
P number of training examples

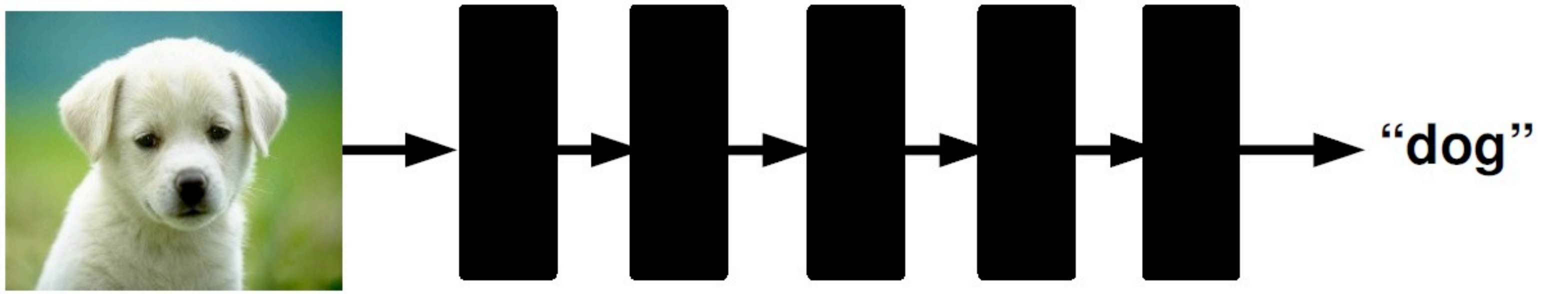
- Goal: predict the target label of unseen inputs

Supervised Learning Examples

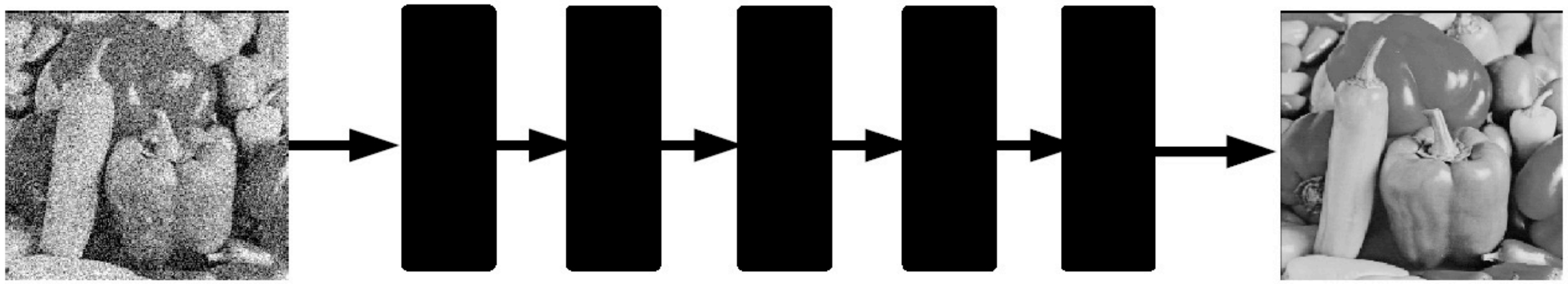


Supervised Deep Learning

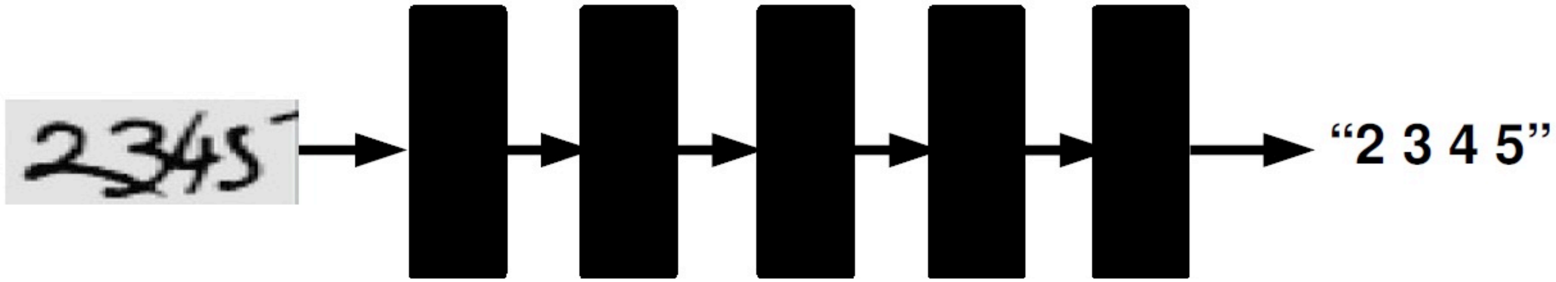
Classification



Denoising



OCR



Neural Networks

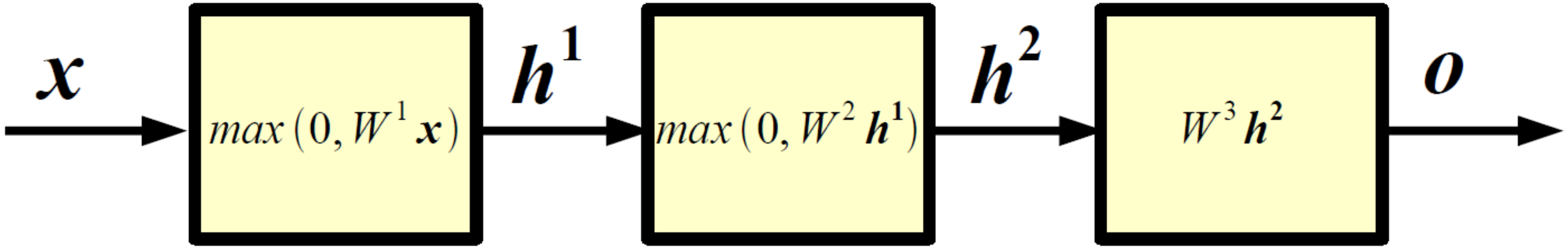
Assumptions (for the next few slides):

- 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



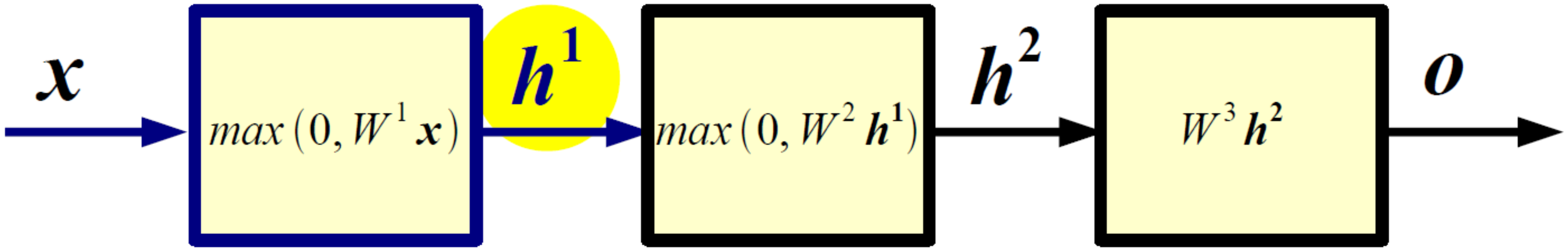
x input
h^1 1-st layer hidden units
h^2 2-nd layer hidden units
o output

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

Forward Propagation

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

Forward Propagation



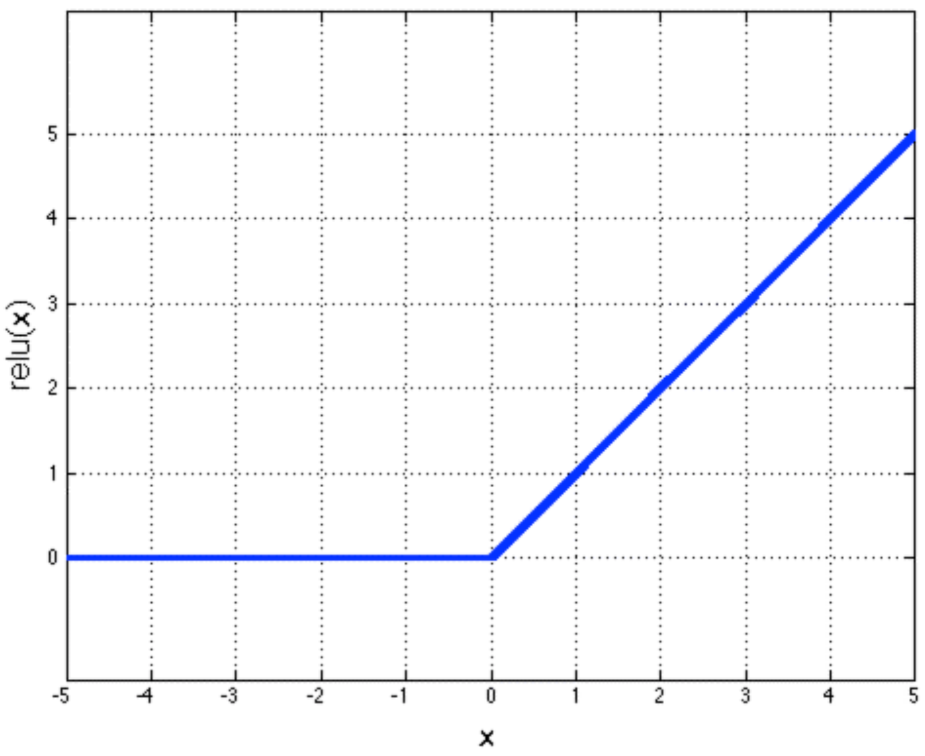
$\boldsymbol{x} \in R^D \quad W^1 \in R^{N_1 \times D} \quad \boldsymbol{b}^1 \in R^{N_1} \quad \boldsymbol{h}^1 \in R^{N_1}$

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

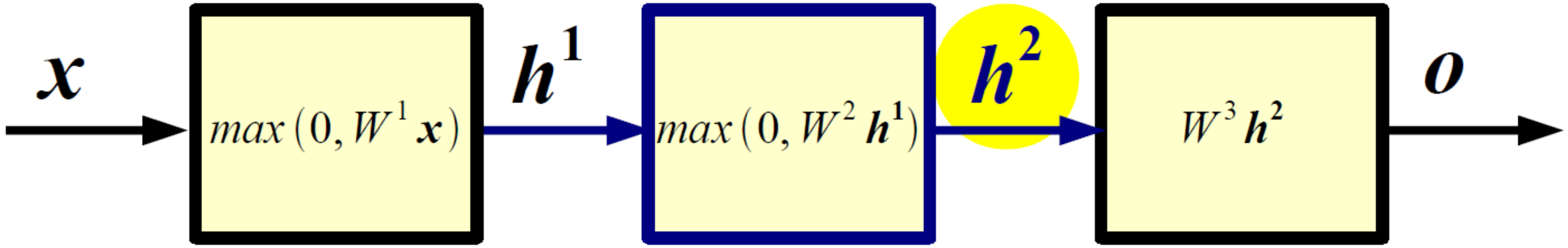
W^1 1^{st} layer weight matrix or weights
$\boldsymbol{b}^1$ 1^{st} layer biases

- 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"**

Rectified Linear Unit (ReLU)



Forward Propagation

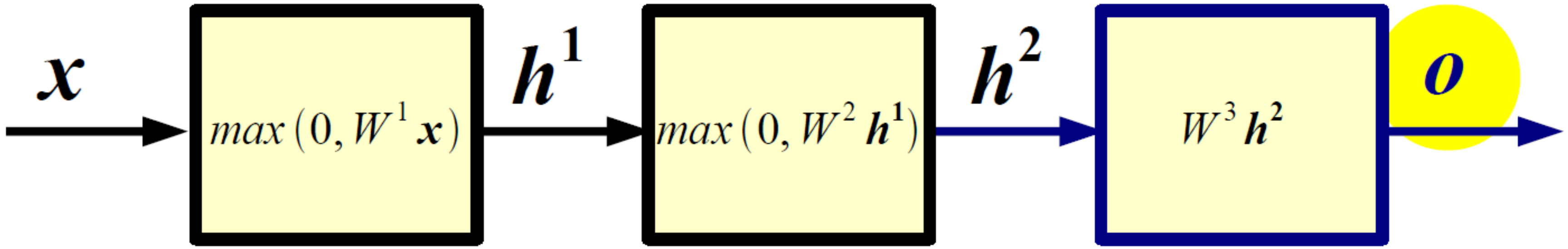


$\boldsymbol{h}^1 \in R^{N_1}$ $\quad W^2 \in R^{N_2 \times N_1}$ $\quad \boldsymbol{b}^2 \in R^{N_2}$ $\quad \boldsymbol{h}^2 \in R^{N_2}$

$$\boldsymbol{h}^2 = max(0, W^2 \boldsymbol{h}^1 + \boldsymbol{b}^2)$$

W^2 2nd layer weight matrix or weights
$\boldsymbol{b}^2$ 2nd layer biases

Forward Propagation

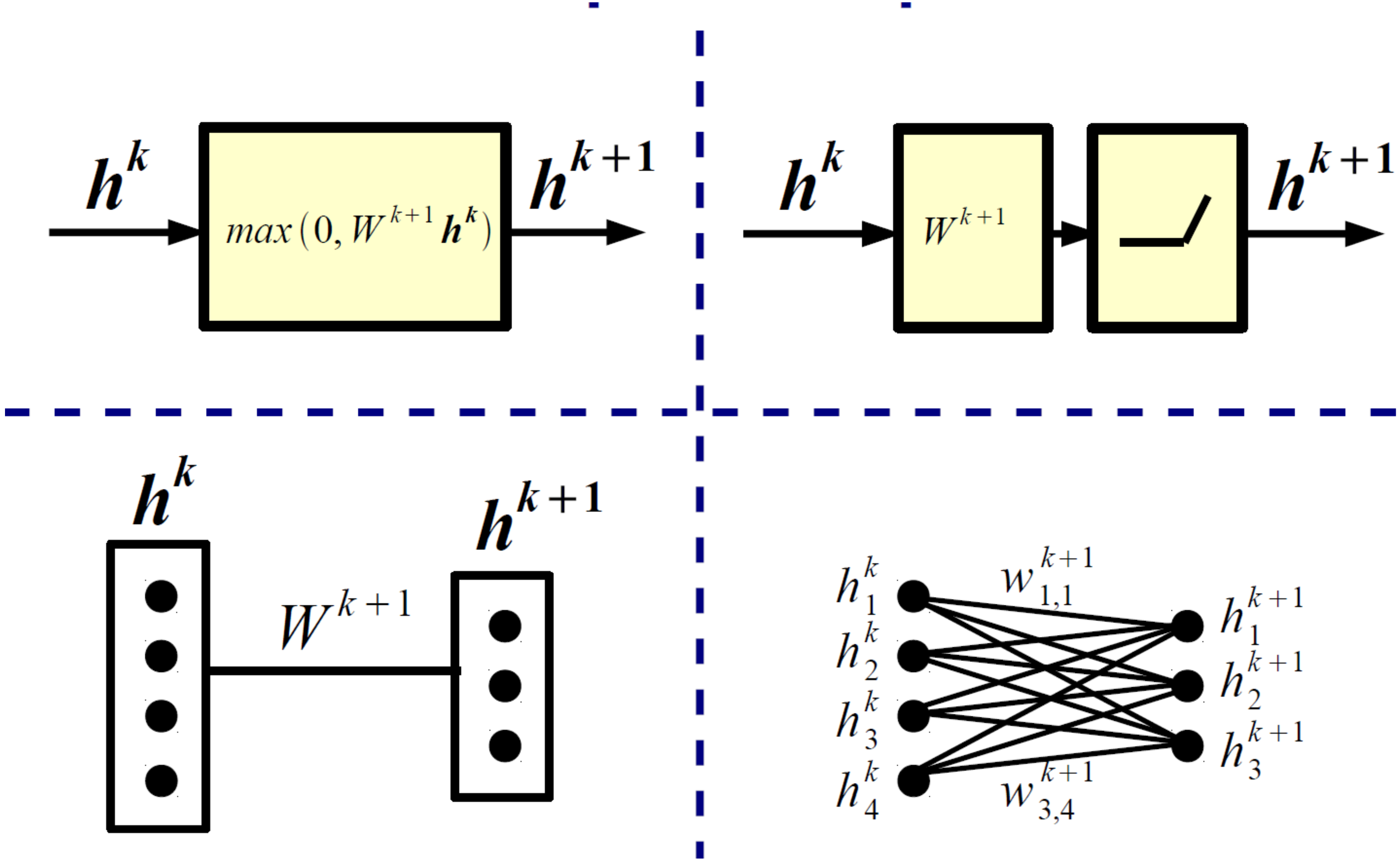


$$h^2 \in R^{N_2} \quad W^3 \in R^{N_3 \times N_2} \quad b^3 \in R^{N_3} \qquad o \in R^{N_3}$$

$$o = max(0, W^3 h^2 + b^3)$$

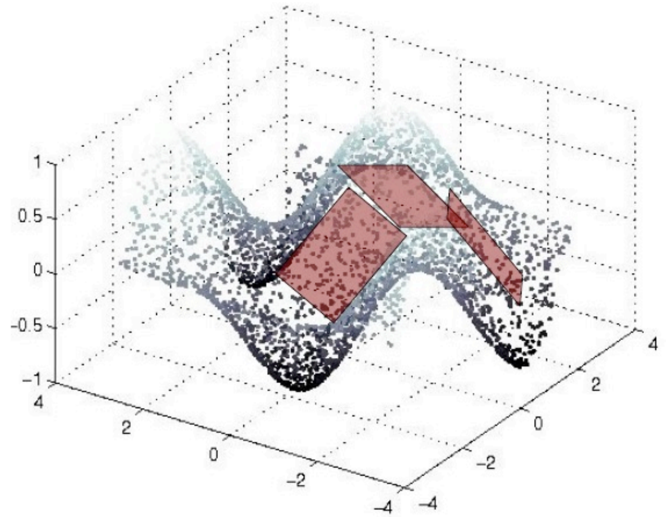
W^3 3rd layer weight matrix or weights
b^3 3rd layer biases

Alternative Graphical Representations



Interpretation

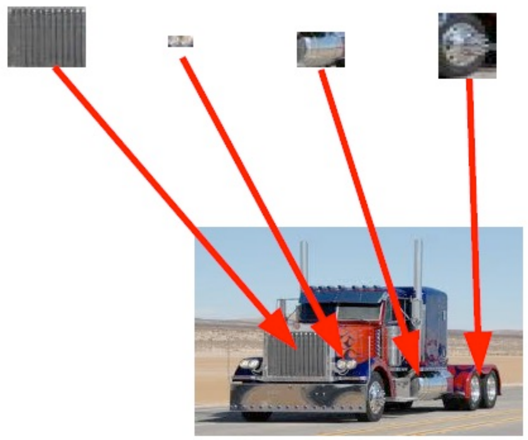
- Question: Why can't the mapping between layers be linear?
- Answer: Because composition of linear functions is a linear function. Neural network would reduce to (1 layer) logistic regression.
- Question: What do ReLU layers accomplish?
- Answer: Piece-wise linear tiling: mapping is locally linear.



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.

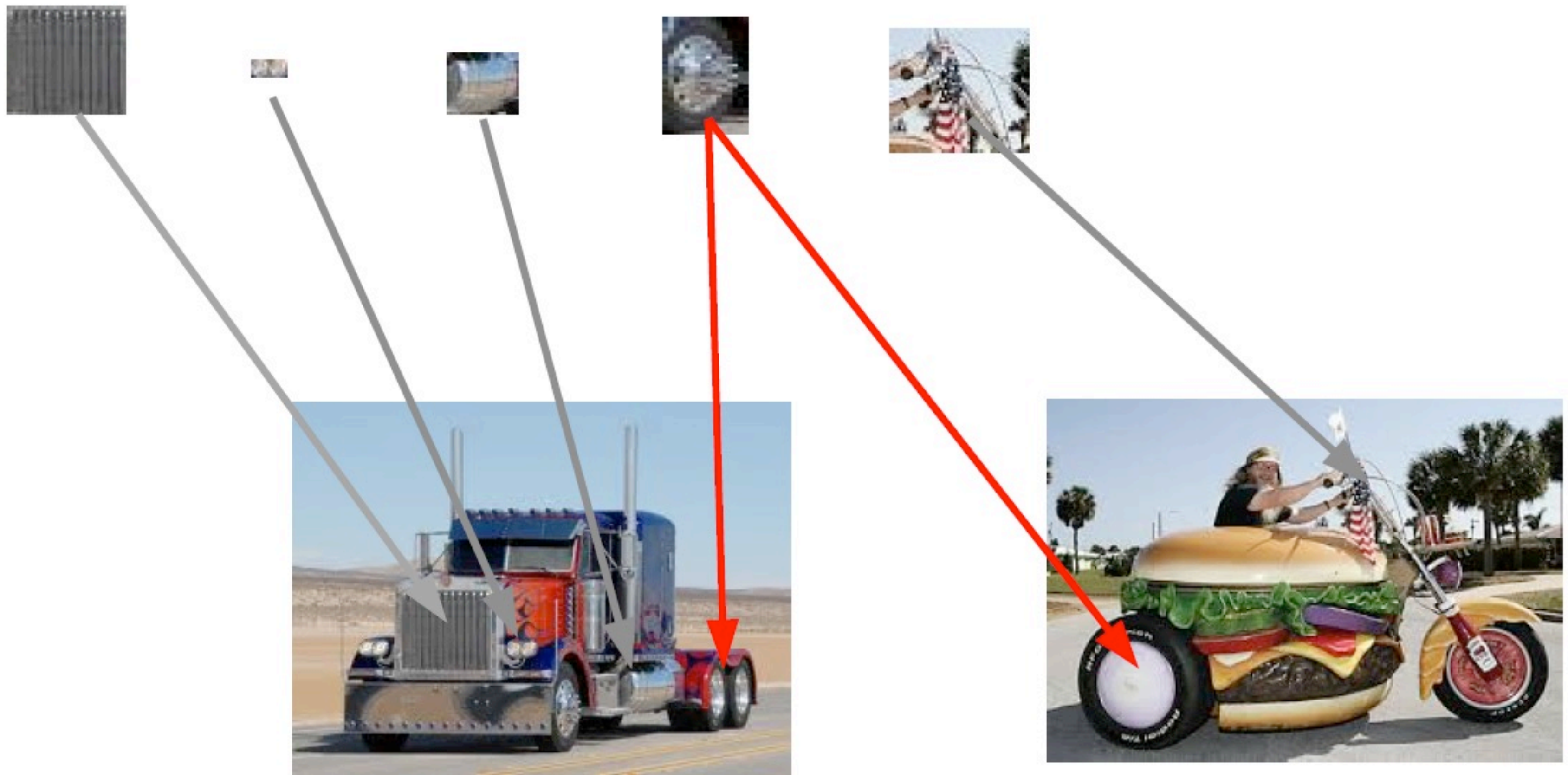
[0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 …] truck feature



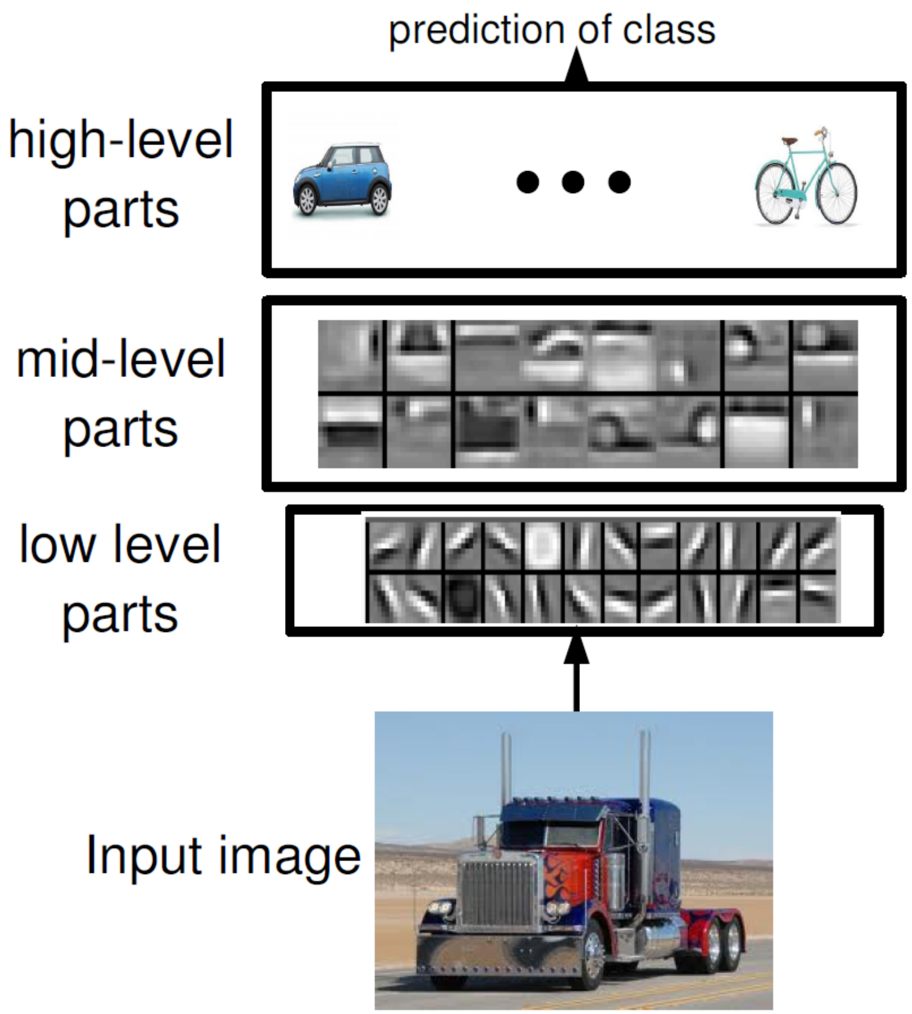
Interpretation

[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

Interpretation

Question: What does a hidden unit do?

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

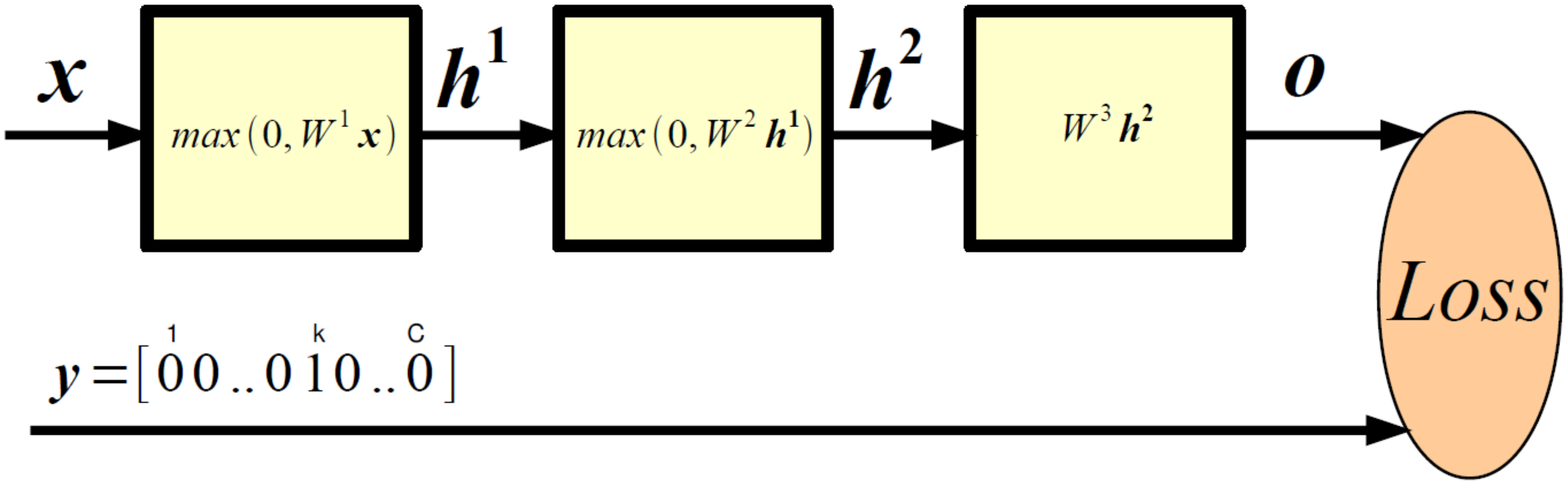
Question: How many layers? How many hidden units?

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

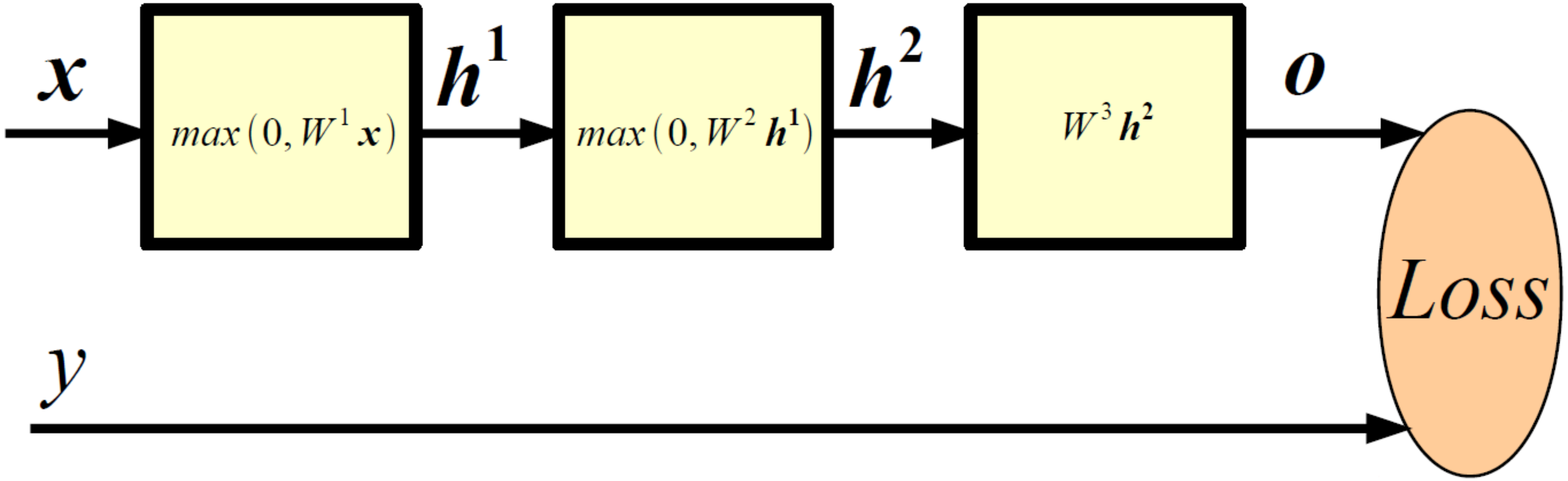
- Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

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

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.

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 ϵ=1e-6 and compute:

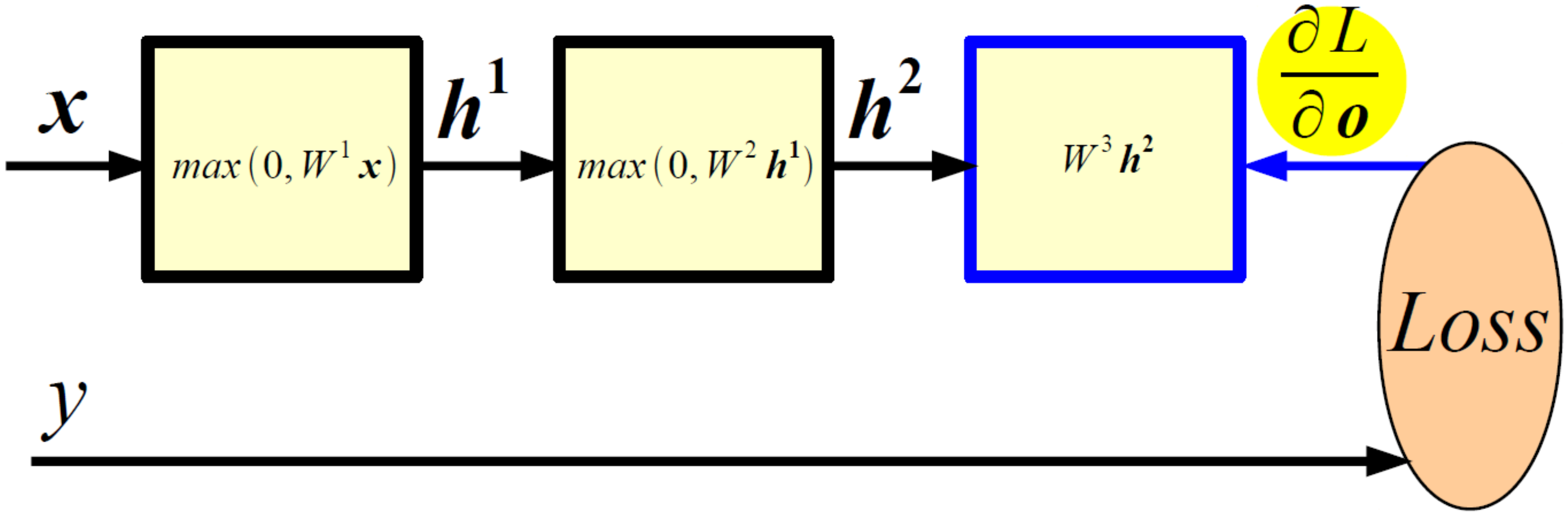
$$L(\boldsymbol{x}, y; \boldsymbol{\theta})$$

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

- Then update:

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

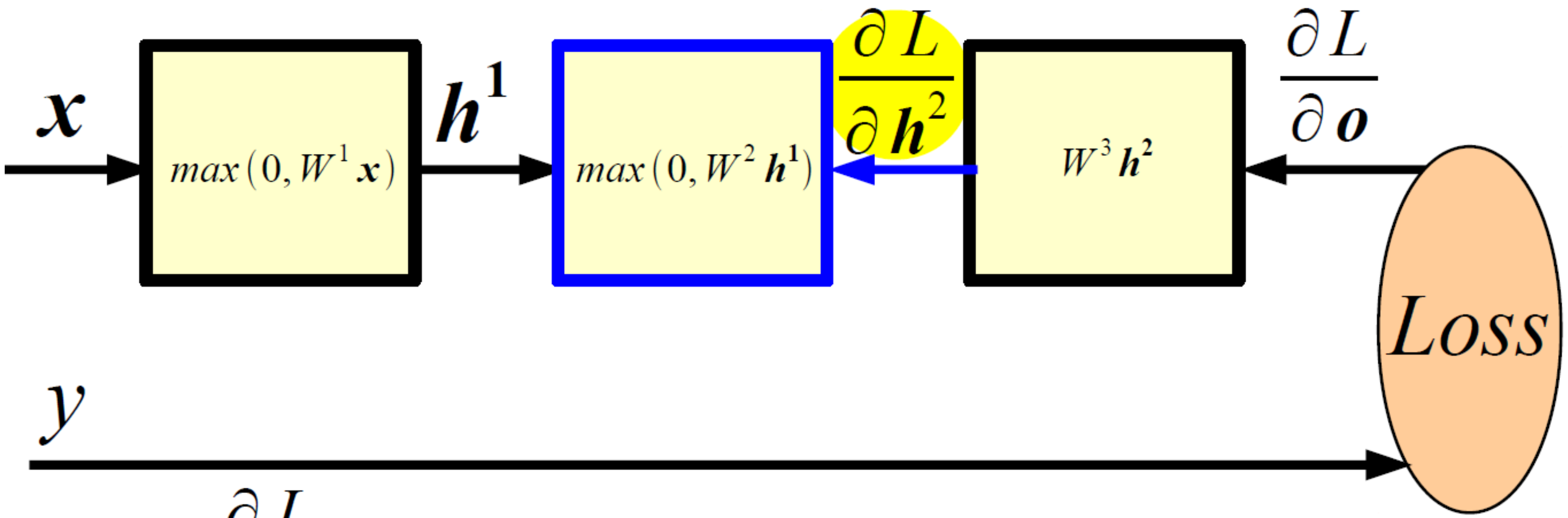
Backward Propagation



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

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

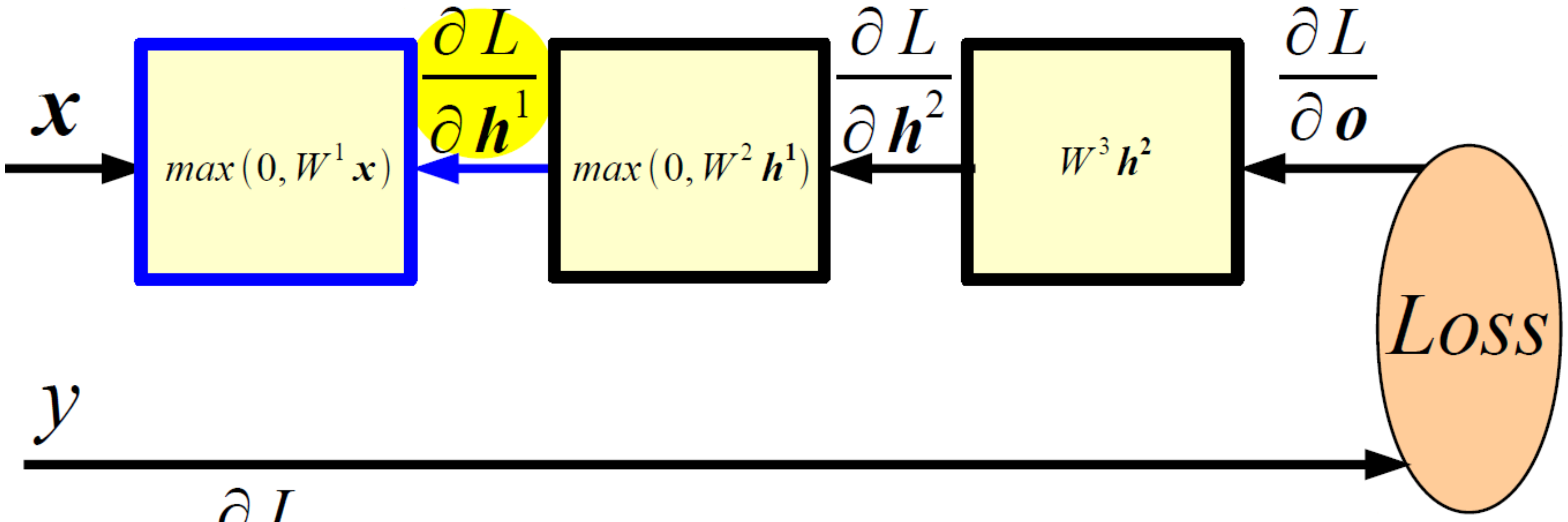
Backward Propagation



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

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

Backward Propagation



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

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

Optimization

Stochastic Gradient Descent

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

Or one of its many variants