Machine Learning Crash Course

Photo: CMU Machine Learning Department protests G20

Computer Vision
James Hays

Slides: Isabelle Guyon, Erik Sudderth, Mark Johnson, Derek Hoiem

Machine Learning Problems

	Supervised Learning	*Unsupervised Learning*
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

The machine learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

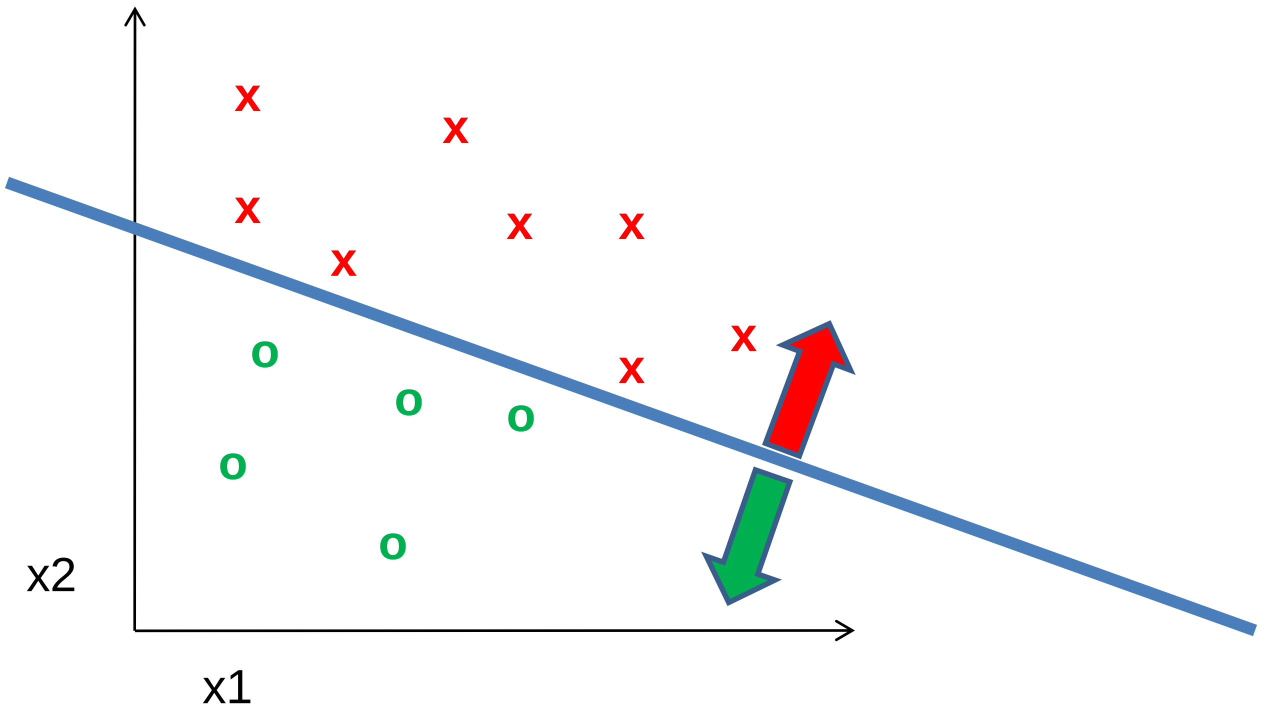
f() = “apple”

f() = “tomato”

f() = “cow”

Learning a classifier

Given some set of features with corresponding labels, learn a function to predict the labels from the features



Generalization

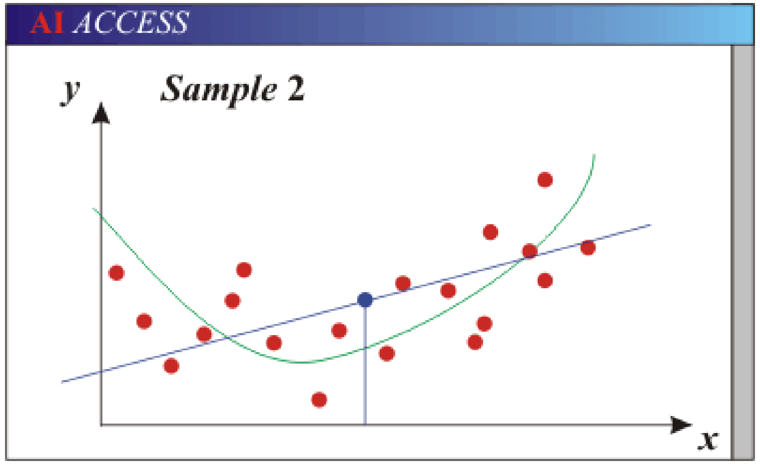
Training set (labels known)

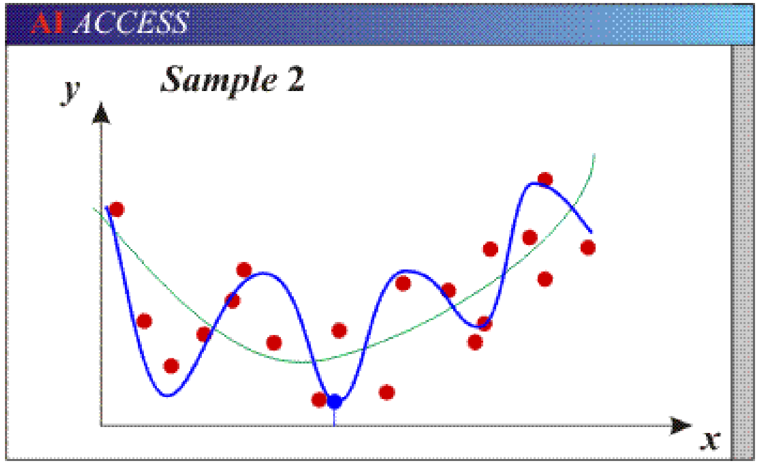
Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

Slide credit: L. Lazebnik

Bias-Variance Trade-off





- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).
- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

- How to reduce variance?
  - Choose a simpler classifier
  - Regularize the parameters
  - Get more training data
- How to reduce bias?
  - Choose a more complex, more expressive classifier
  - Remove regularization
  - (These might not be safe to do unless you get more training data)

Slide credit: D. Hoiem

Very brief tour of some classifiers

- **K-nearest neighbor**
- **SVM**
- Neural networks (separate lecture)
- Boosted Decision Trees
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- RBMs
- Etc.

Generative vs. Discriminative Classifiers

Generative Models

- Represent both the data and the labels
- Often, makes use of conditional independence and priors
- Examples
  - Naïve Bayes classifier
  - Bayesian network

- Models of data may apply to future prediction problems

Discriminative Models

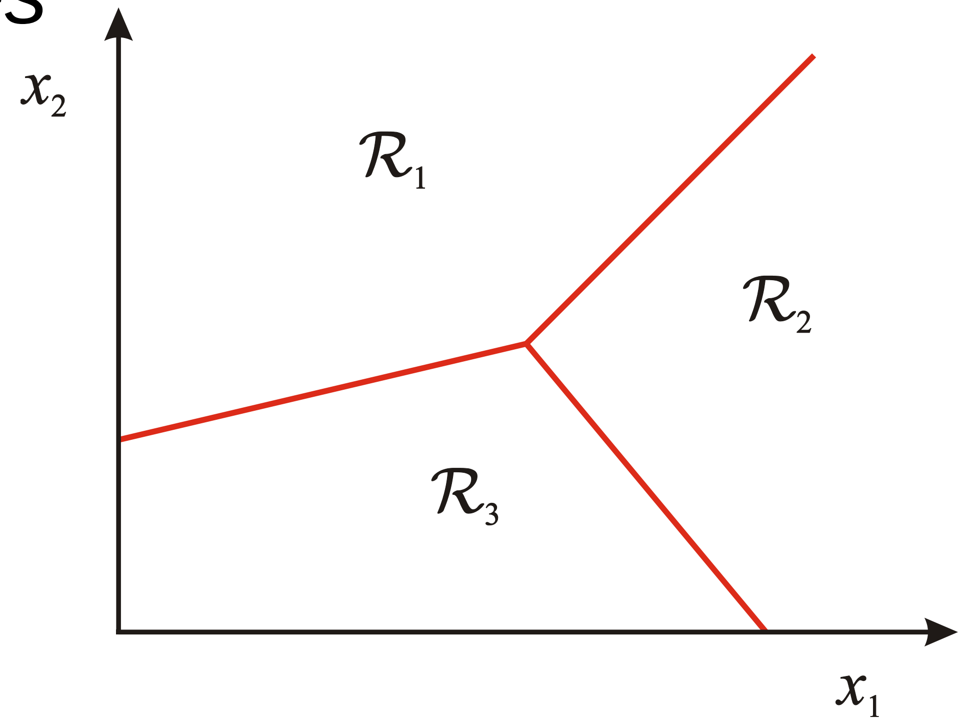
- Learn to directly predict the labels from the data
- Often, assume a simple boundary (e.g., linear)
- Examples
  - Logistic regression
  - SVM
  - Boosted decision trees

- Often easier to predict a label from the data than to model the data

Slide credit: D. Hoiem

Classification

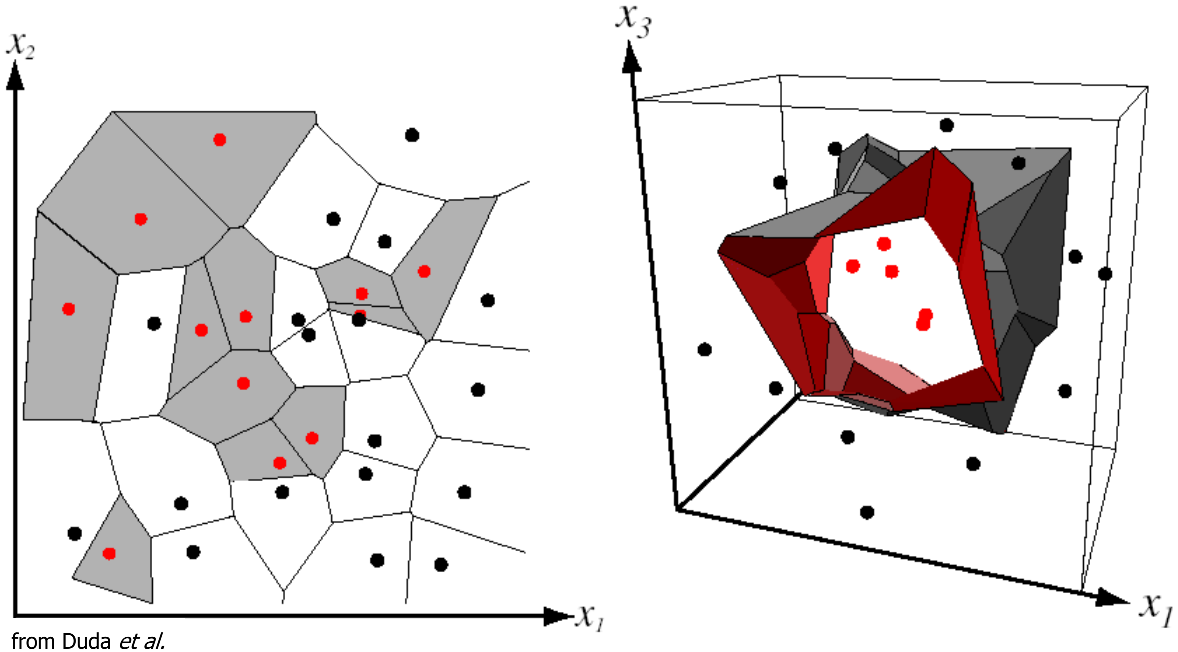
- Assign input vector to one of two or more classes
- Any decision rule divides input space into *decision regions* separated by *decision boundaries*



Slide credit: L. Lazebnik

Nearest Neighbor Classifier

- Assign label of nearest training data point to each test data point

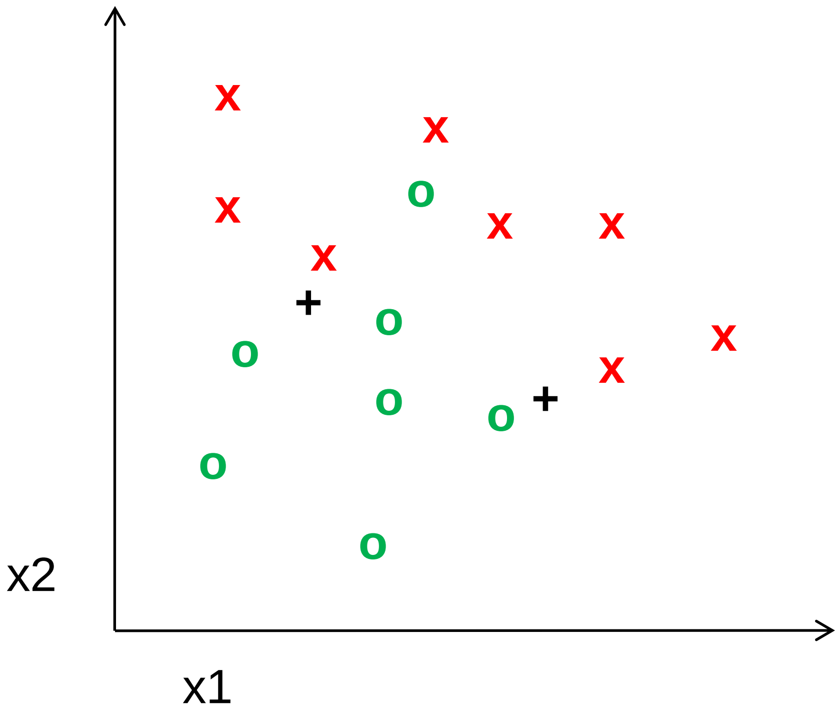


from Duda *et al.*

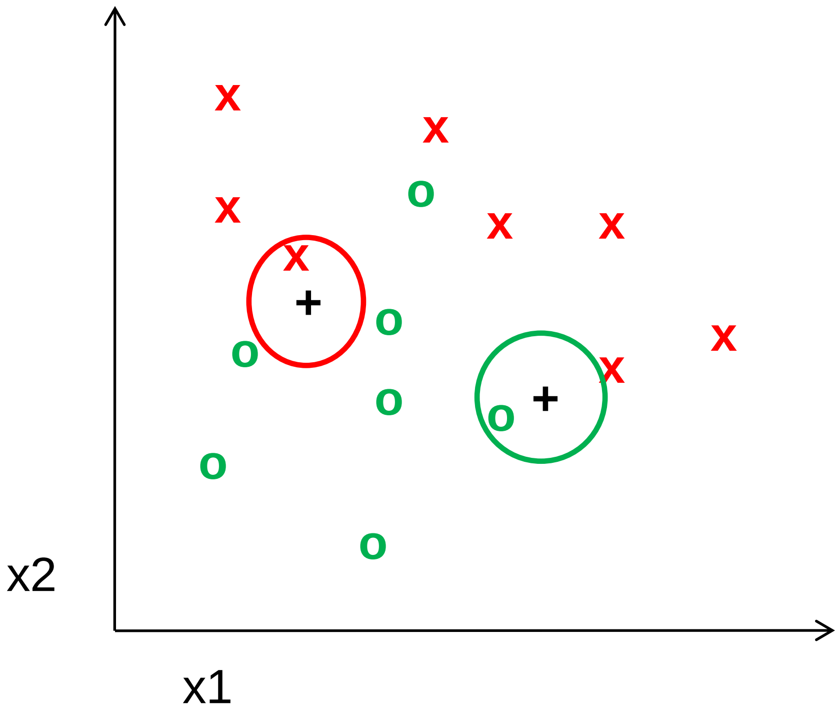
Voronoi partitioning of feature space for two-category 2D and 3D data

Source: D. Lowe

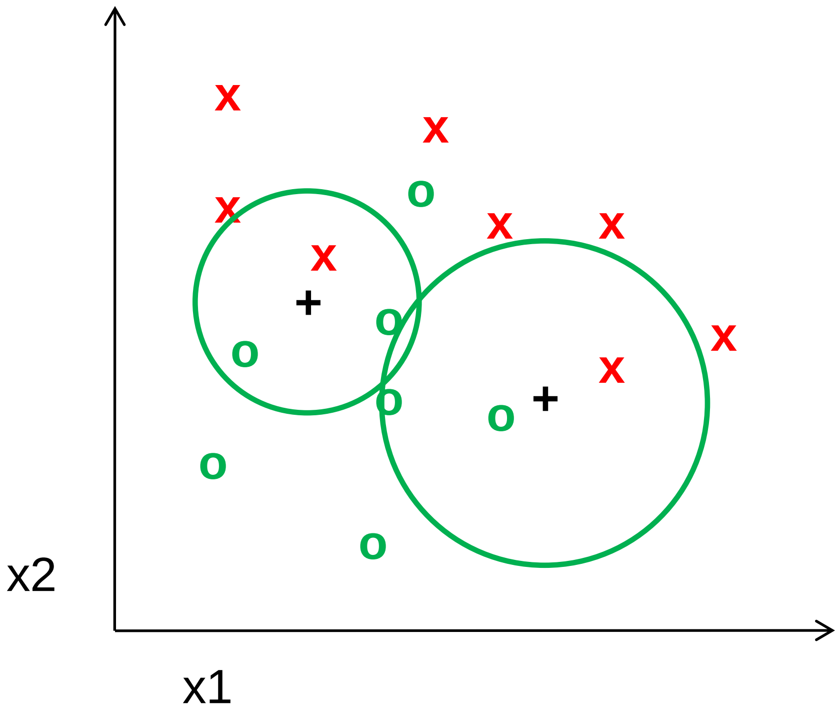
K-nearest neighbor



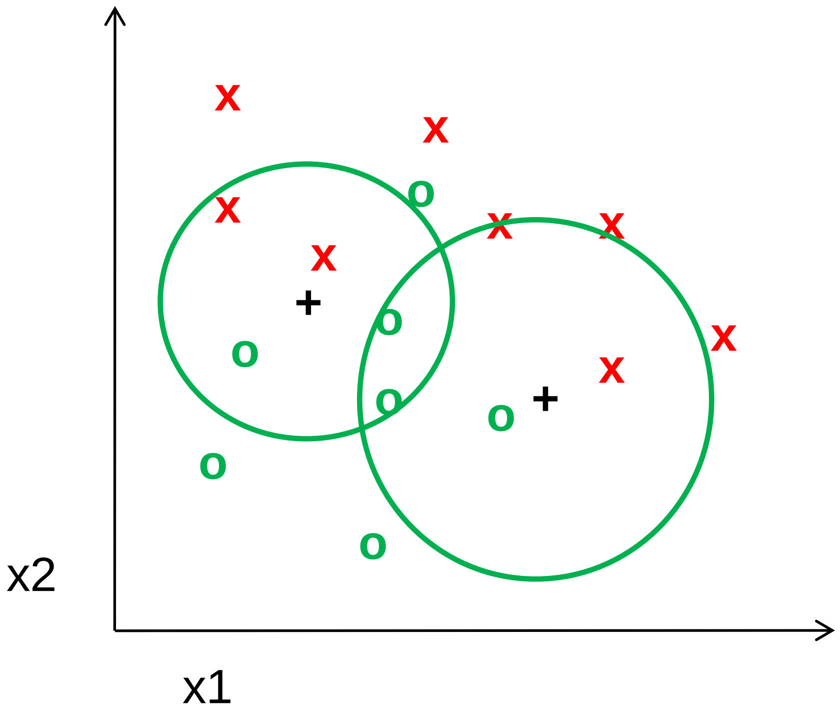
1-nearest neighbor



3-nearest neighbor



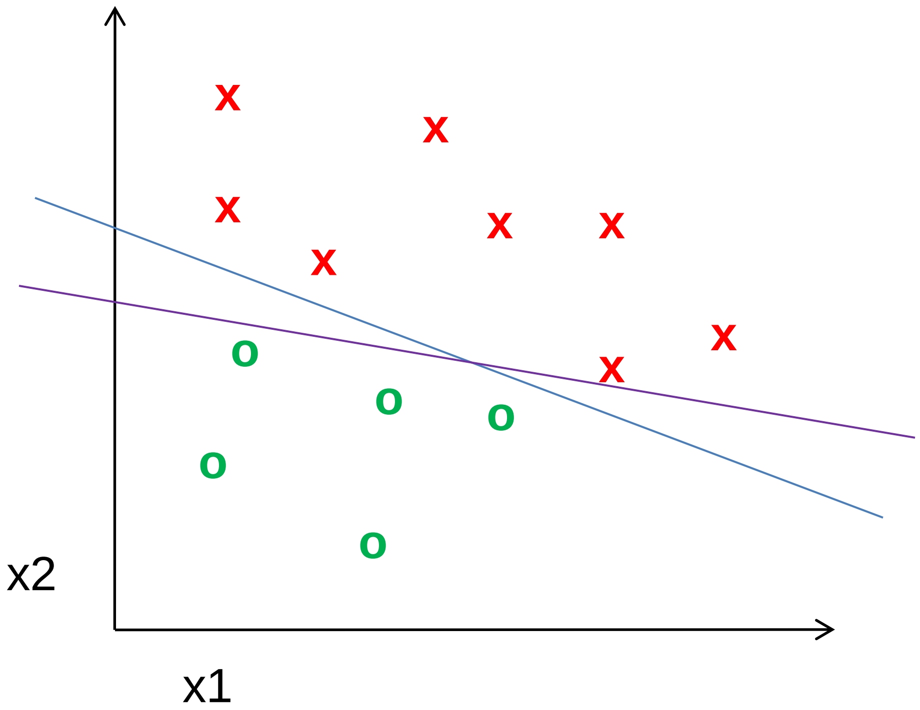
5-nearest neighbor



Using K-NN

- Simple, a good one to try first
- With infinite examples, 1-NN provably has error that is at most twice Bayes optimal error

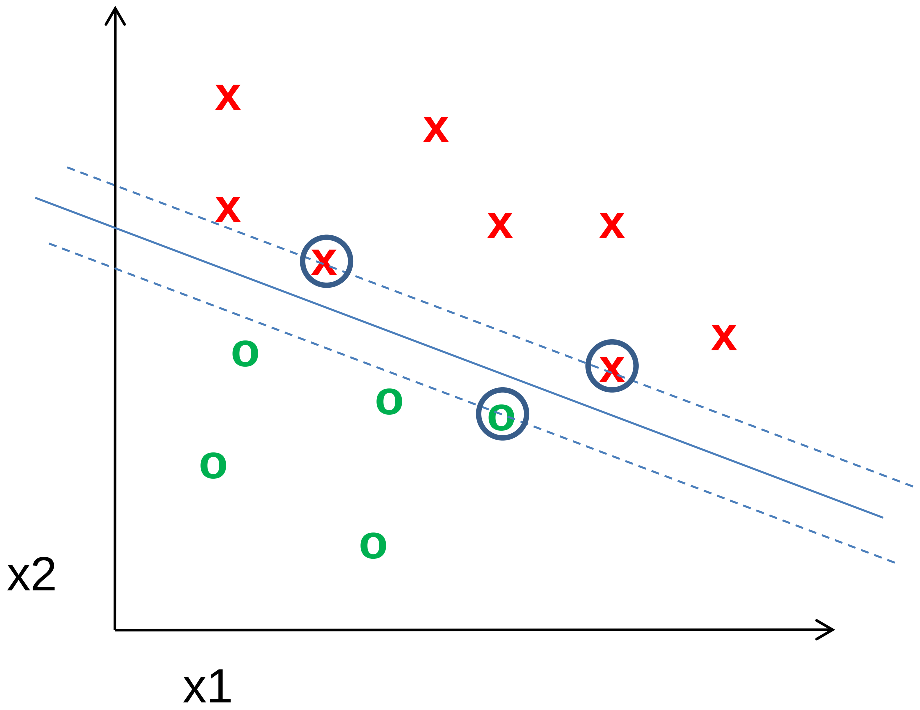
Classifiers: Linear SVM



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

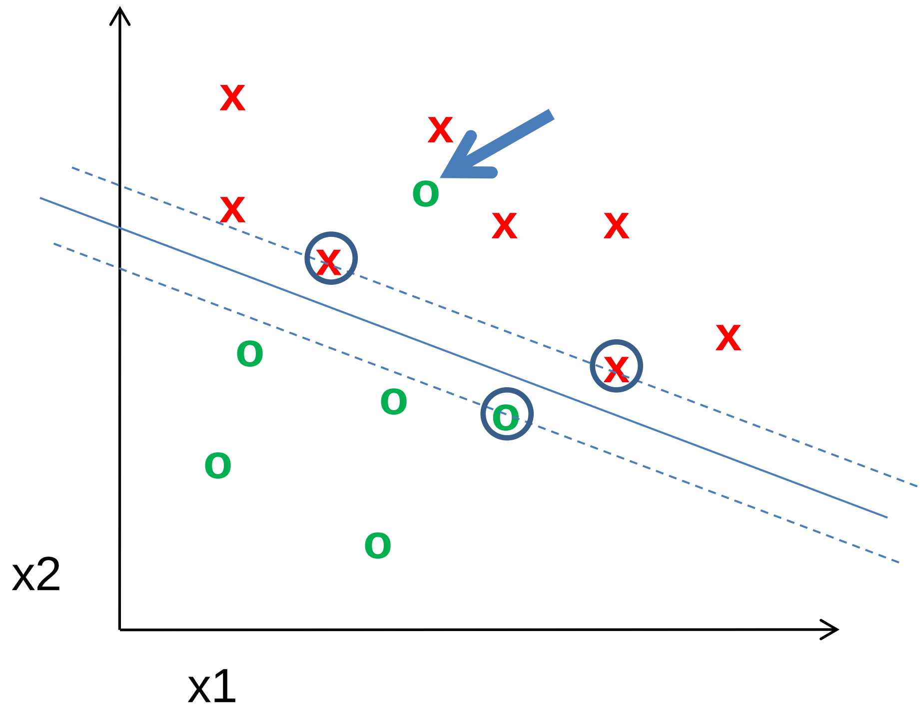
Classifiers: Linear SVM



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Classifiers: Linear SVM



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

What about multi-class SVMs?

- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
  - Traning: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example

Slide credit: L. Lazebnik

SVMs: Pros and cons

- Pros
  - Many publicly available SVM packages: http://www.kernel-machines.org/software
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes

- Cons
  - No “direct” multi-class SVM, must combine two-class SVMs
  - Computation, memory
    - During training time, must compute matrix of kernel values for every pair of examples (for nonlinear SVMs)
    - Learning can take a very long time for large-scale problems

What to remember about classifiers

- No free lunch: machine learning algorithms are tools, not dogmas
- Try simple classifiers first
- Better to have smart features and simple classifiers than simple features and smart classifiers
- Use increasingly powerful classifiers with more training data (bias-variance tradeoff)

Slide credit: D. Hoiem

Machine Learning Considerations

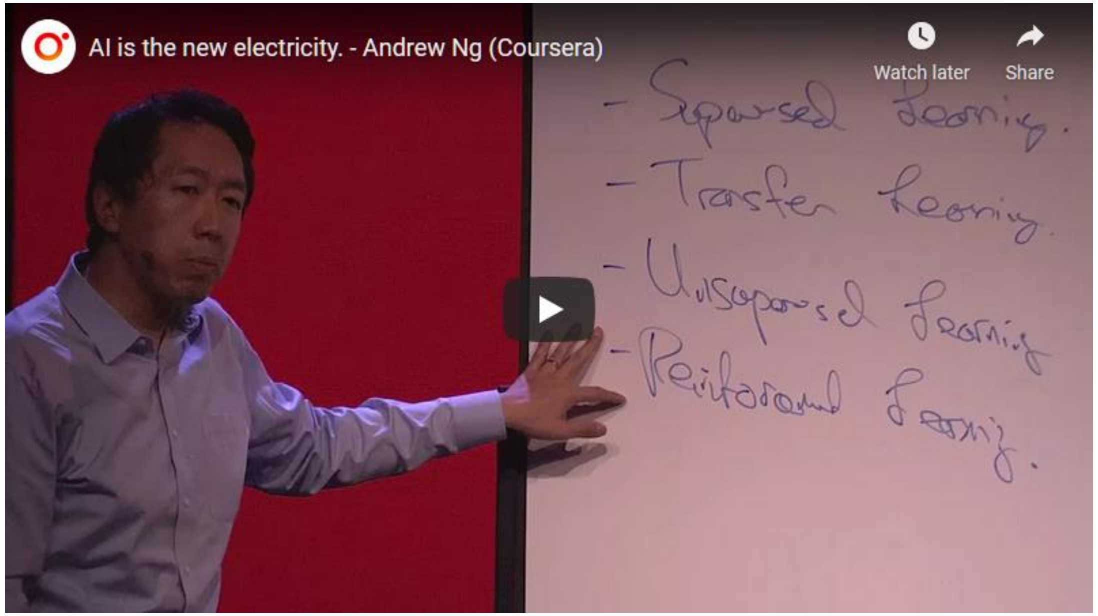
- 3 important design decisions:
  1) What data do I use?
  2) How do I represent my data (what feature)?
  3) What classifier / regressor / machine learning tool do I use?
- These are in decreasing order of importance
- Deep learning addresses 2 and 3 simultaneously (and blurs the boundary between them).
- You can take the representation from deep learning and use it with any classifier.

Machine Learning Problems

	Supervised Learning	***Unsupervised Learning***
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

- Andrew Ng’s ranking of machine learning impact
  1. Supervised Learning
  2. Transfer Learning
  3. Unsupervised Learning* (better described as “self-supervised” learning)
  4. Reinforcement Learning

James thinks 2 and 3 might have switched ranks.




www.grand-illusions.com

Deep Learning
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

$\left\{ (\boldsymbol{x}^i, y^i), i=1 \ldots P \right\}$ 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.

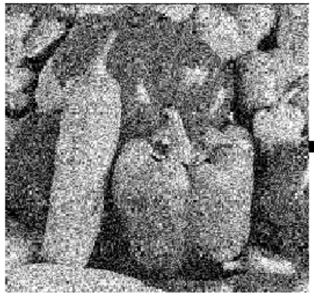
Supervised Learning: Examples

Classification

"dog"

classification

Denoising



regression

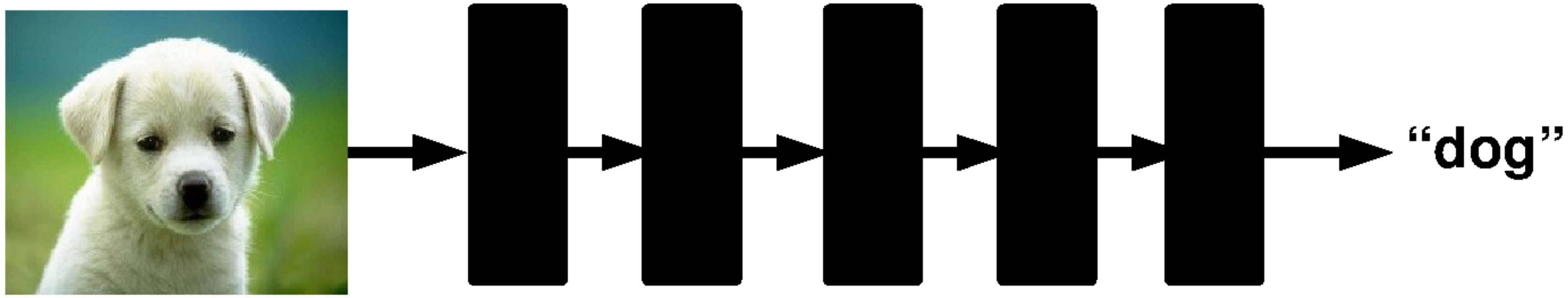
OCR

"2 3 4 5"

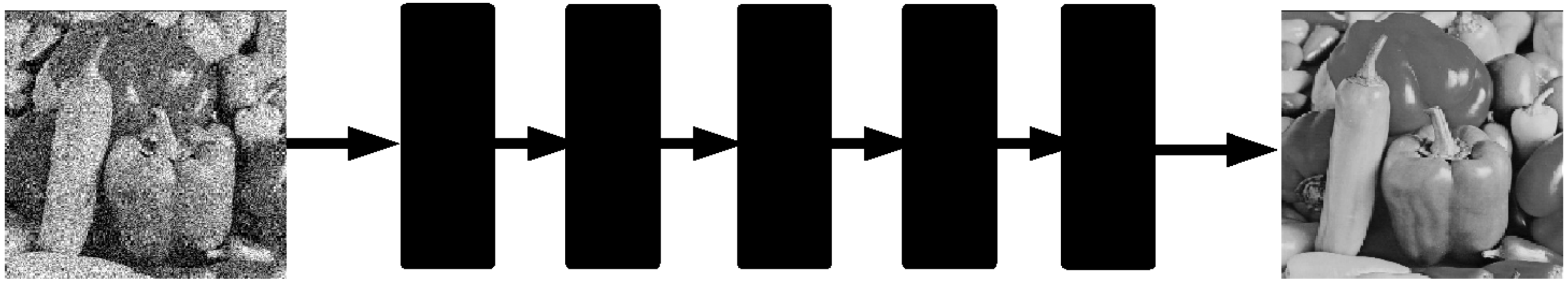
structured prediction

Supervised Deep Learning

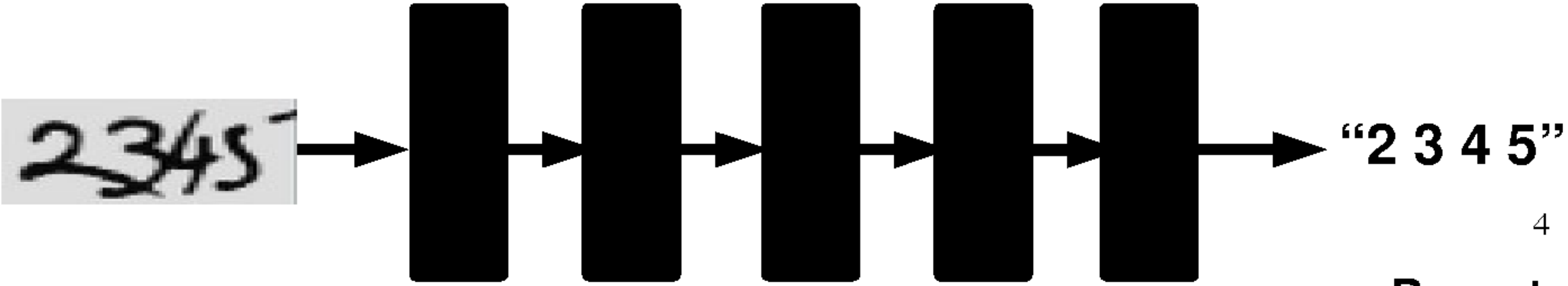
Classification



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)

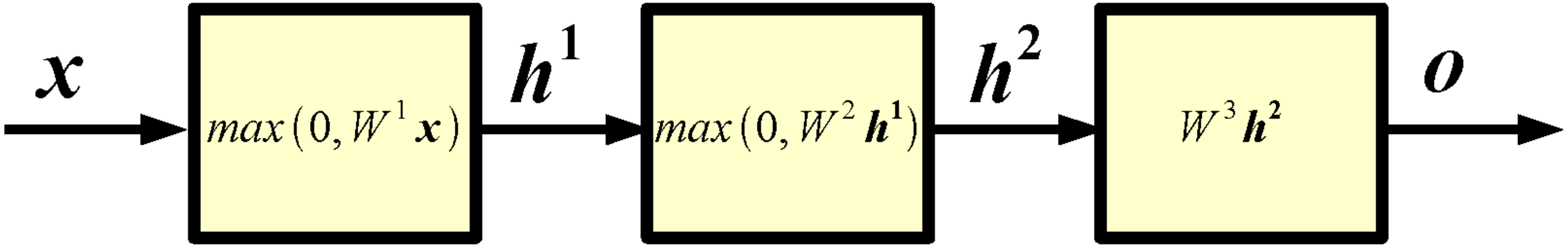
Question: what class of functions shall we consider to map the input into the output?

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



$\boldsymbol{x}$ input

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

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

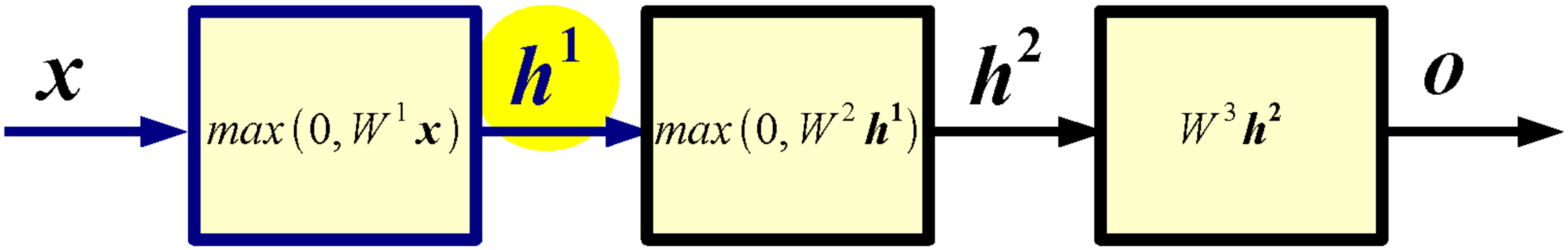
$\boldsymbol{o}$ output

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



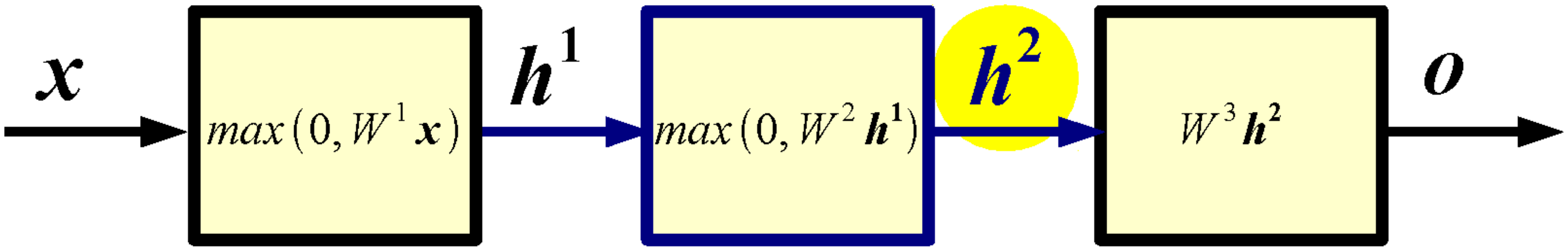
$x \in R^D$ $\quad W^1 \in R^{N_1 \times D}$ $\quad b^1 \in R^{N_1}$ $\quad h^1 \in R^{N_1}$

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

W^1 1-st layer weight matrix or weights
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



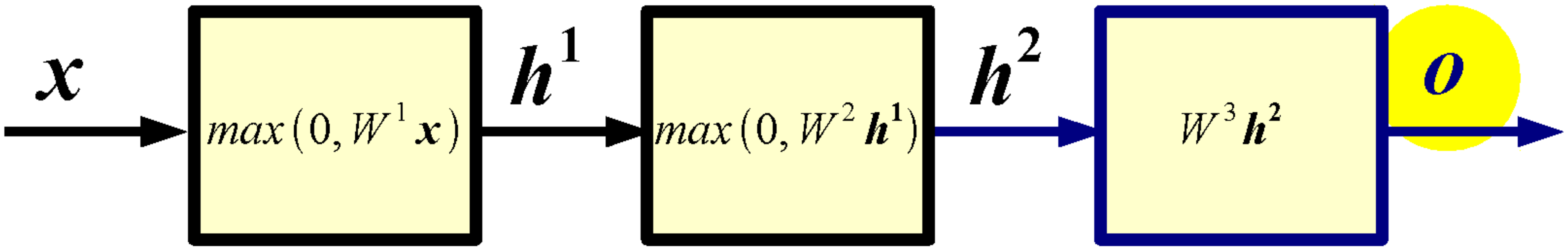
$$h^1 \in R^{N_1} \quad W^2 \in R^{N_2 \times N_1} \quad b^2 \in R^{N_2} \quad h^2 \in R^{N_2}$$

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

W^2 2-nd layer weight matrix or weights

b^2 2-nd layer biases

Forward Propagation

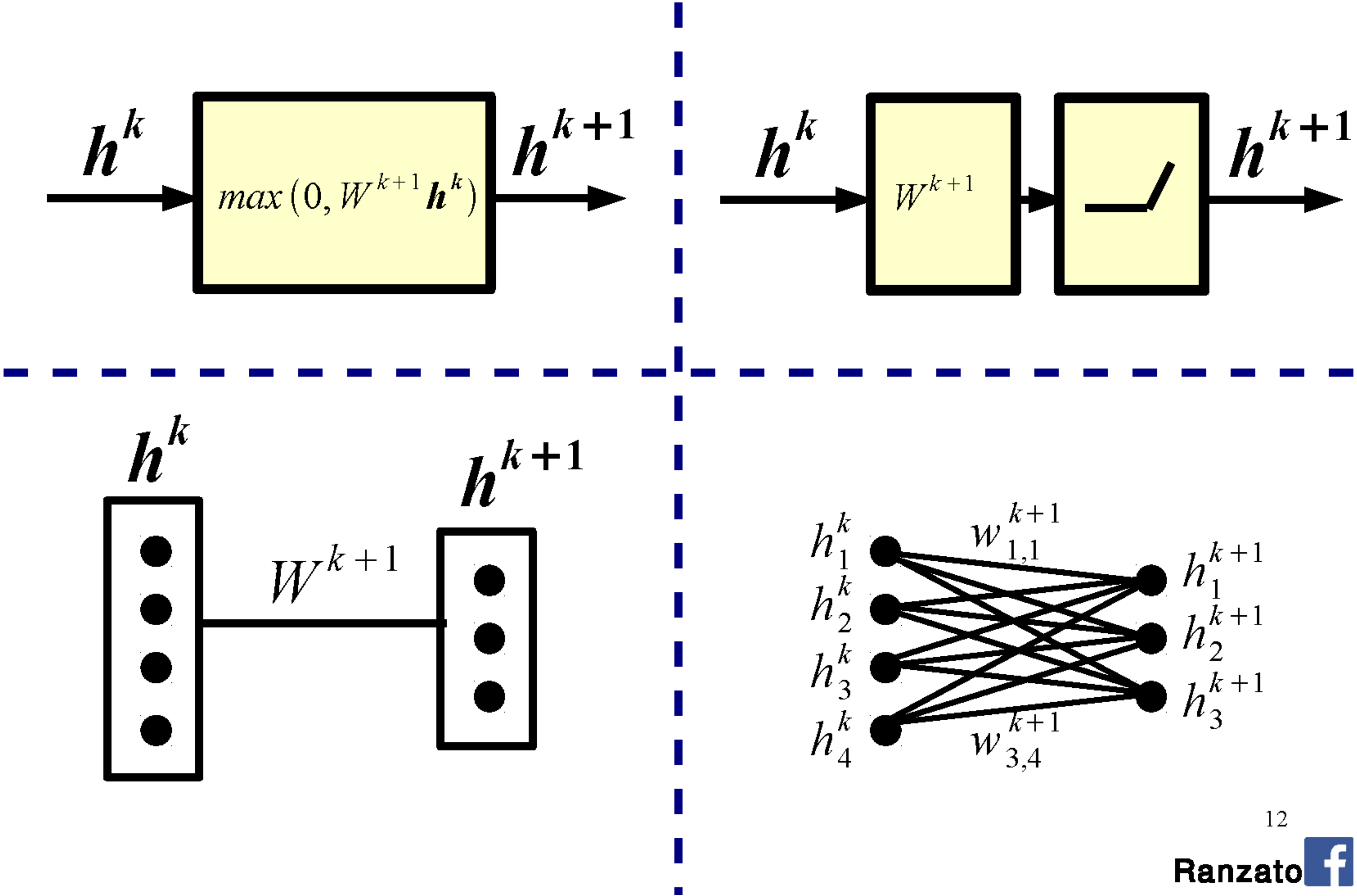


$\boldsymbol{h}^2 \in R^{N_2} \quad W^3 \in R^{N_3 \times N_2} \quad \boldsymbol{b}^3 \in R^{N_3} \qquad \boldsymbol{o} \in R^{N_3}$

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

W^3 3-rd layer weight matrix or weights
$\boldsymbol{b}^3$ 3-rd layer biases

Alternative Graphical Representation

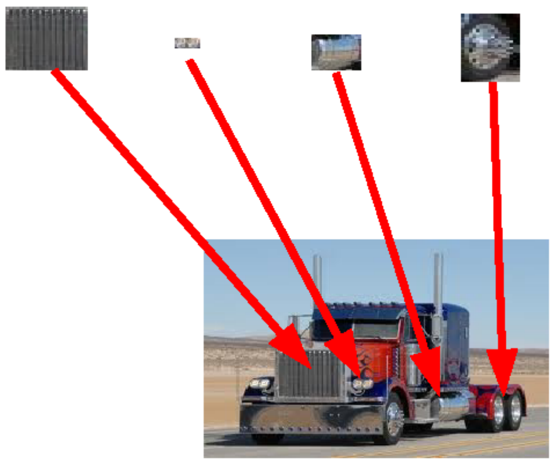


Interpretation

Question: Why 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

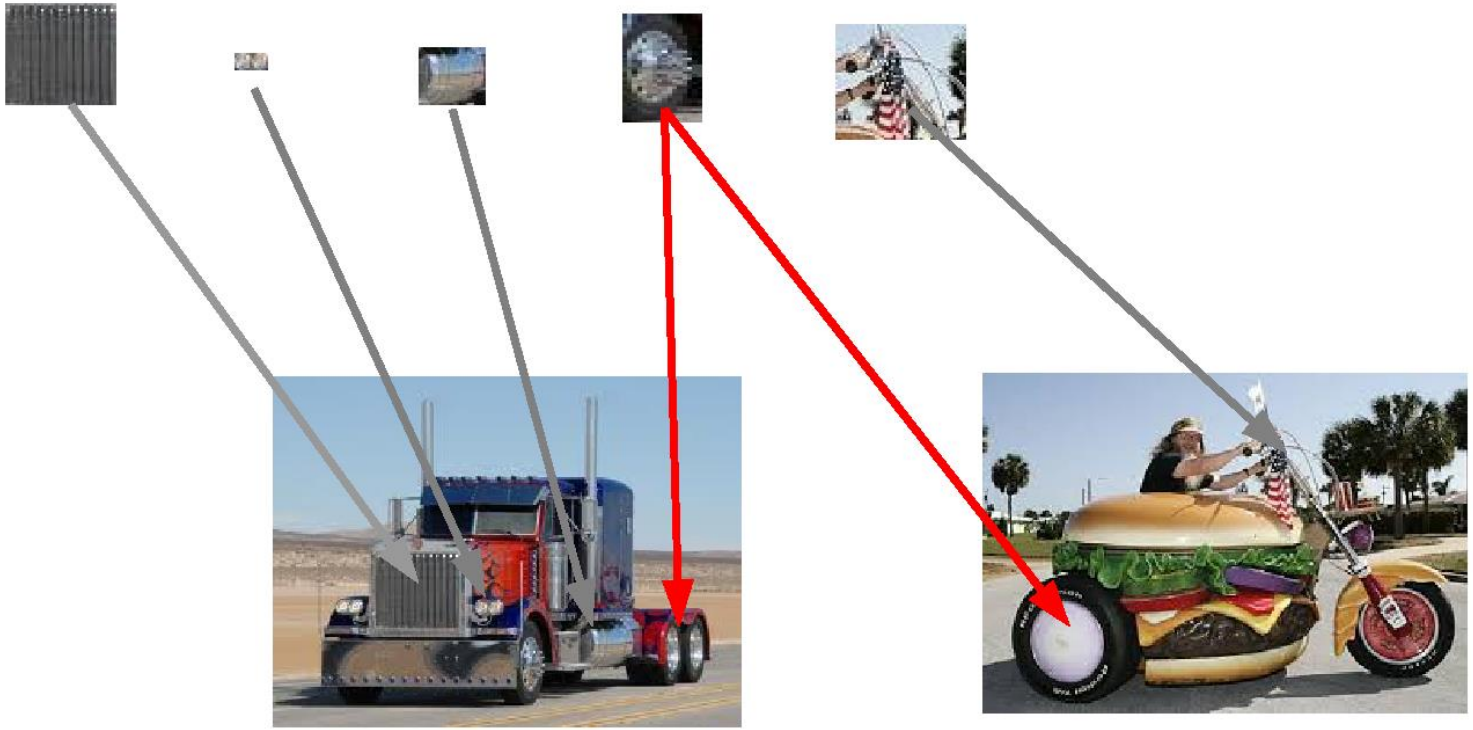


Exponentially more efficient than a 1-of-N representation (a la k-means)

Interpretation

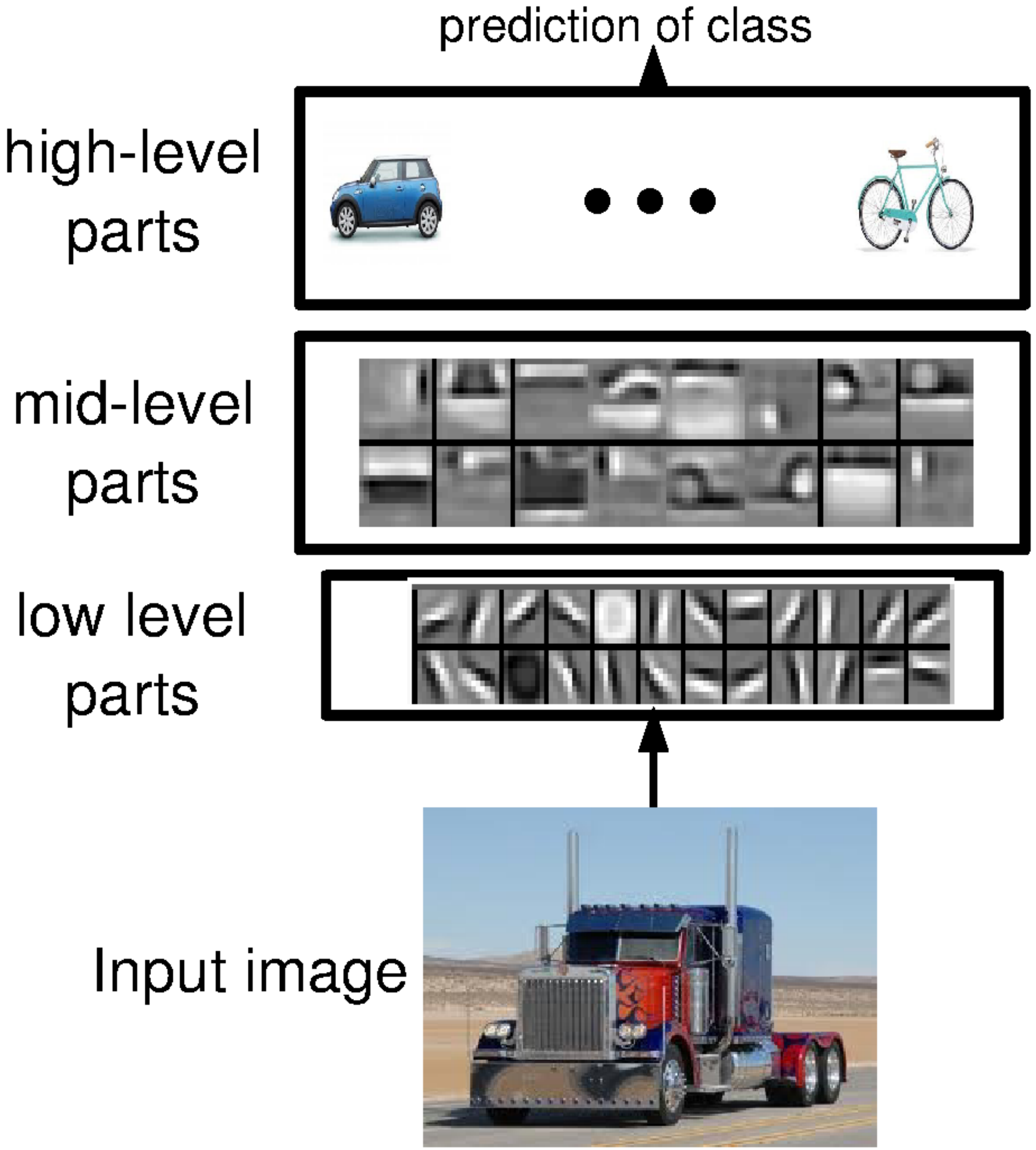
[1 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1…] motorbike

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



Ranzato

Interpretation



- distributed representations
- feature sharing
- compositionality

Lee et al. "Convolutional DBN's ..." ICML 2009

Ranzato

Interpretation

Question: What does a hidden unit do?

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

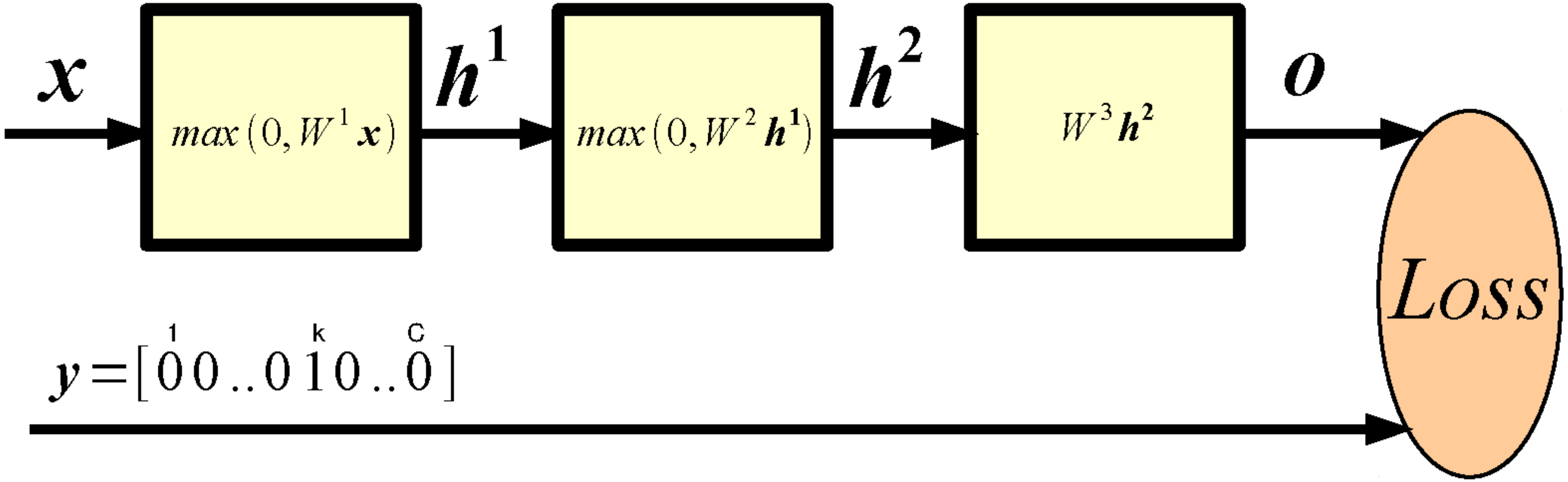
Question: How many layers? How many hidden units?

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.

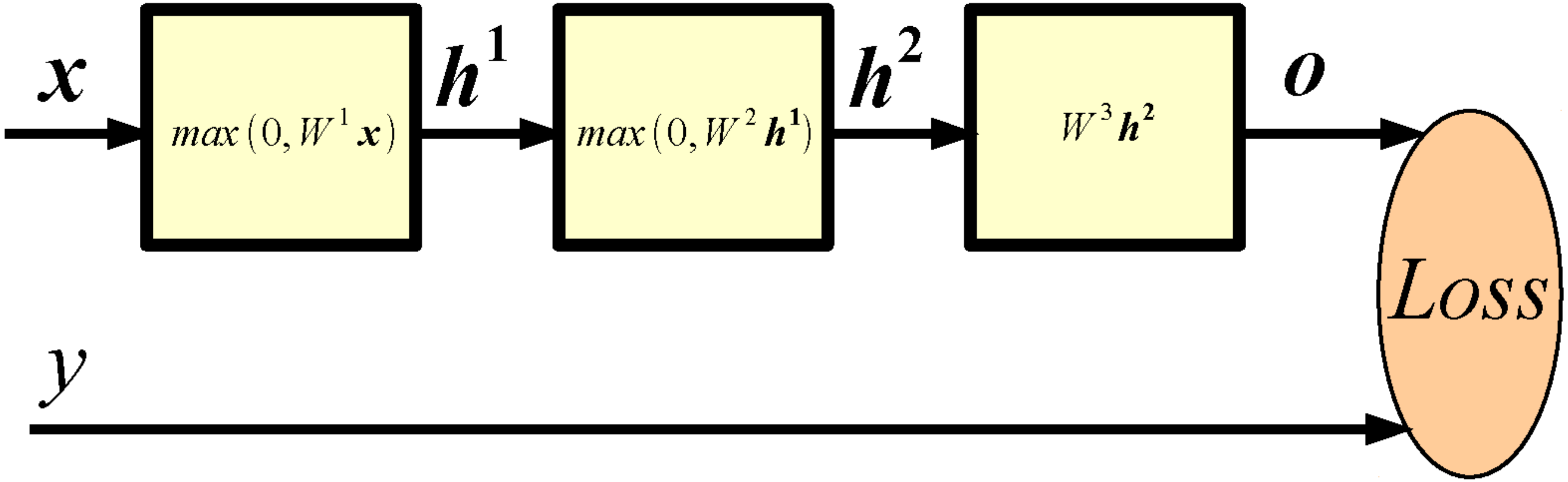
$$\boldsymbol{\theta}^* = arg\ min_{\boldsymbol{\theta}} \sum_{n=1}^{P} L(\boldsymbol{x}^n, y^n; \boldsymbol{\theta})$$

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:

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

Ranzato

Derivative w.r.t. Input of Softmax

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

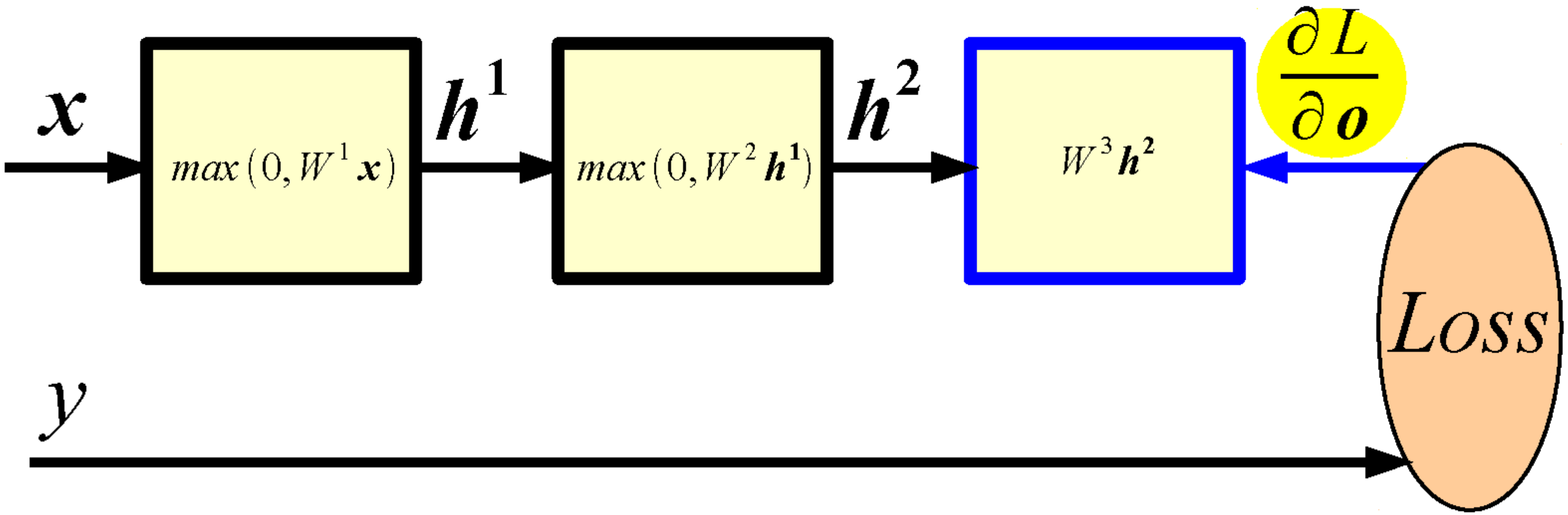
$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j | \boldsymbol{x}) \qquad \boldsymbol{y} = [\overset{1}{0}\, 0 \,..\, 0\, \overset{k}{1}\, 0 \,..\, \overset{C}{0}]$$

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

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

Ranzato

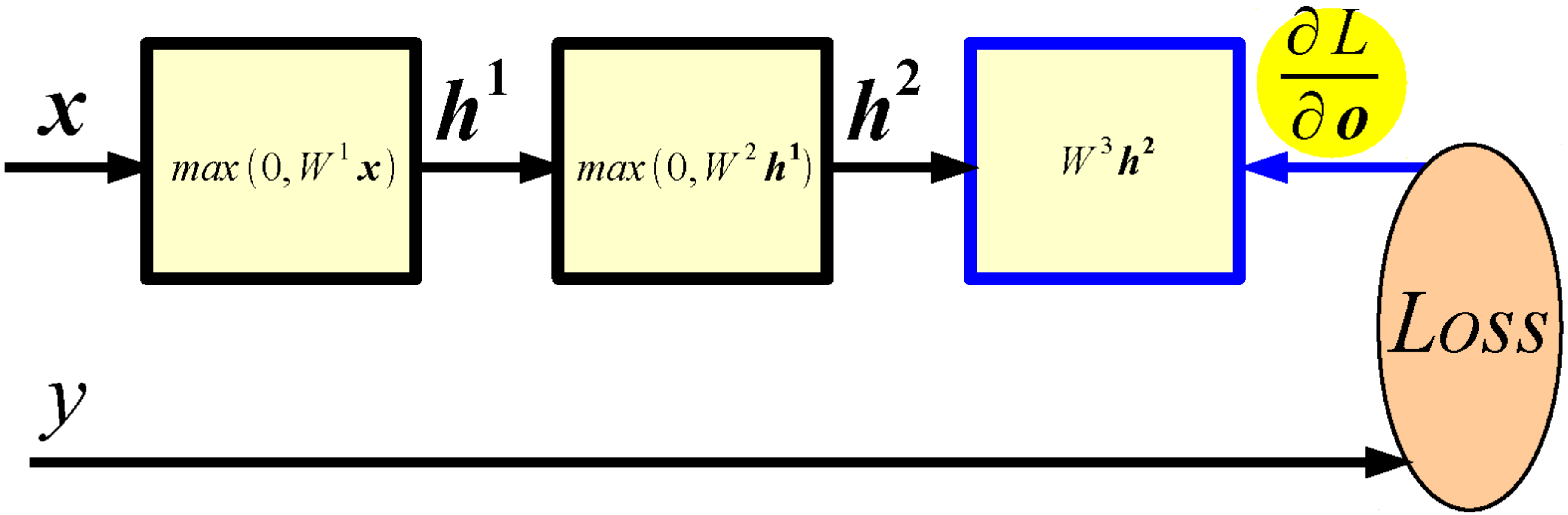
Backward Propagation



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

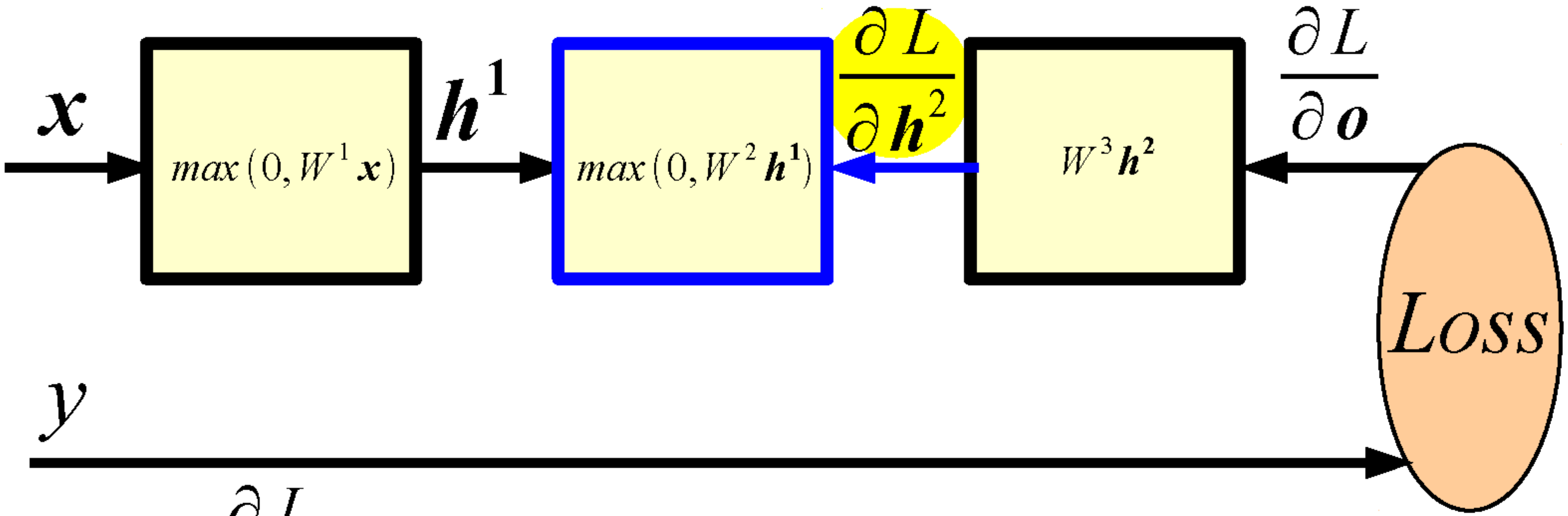


Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

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

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

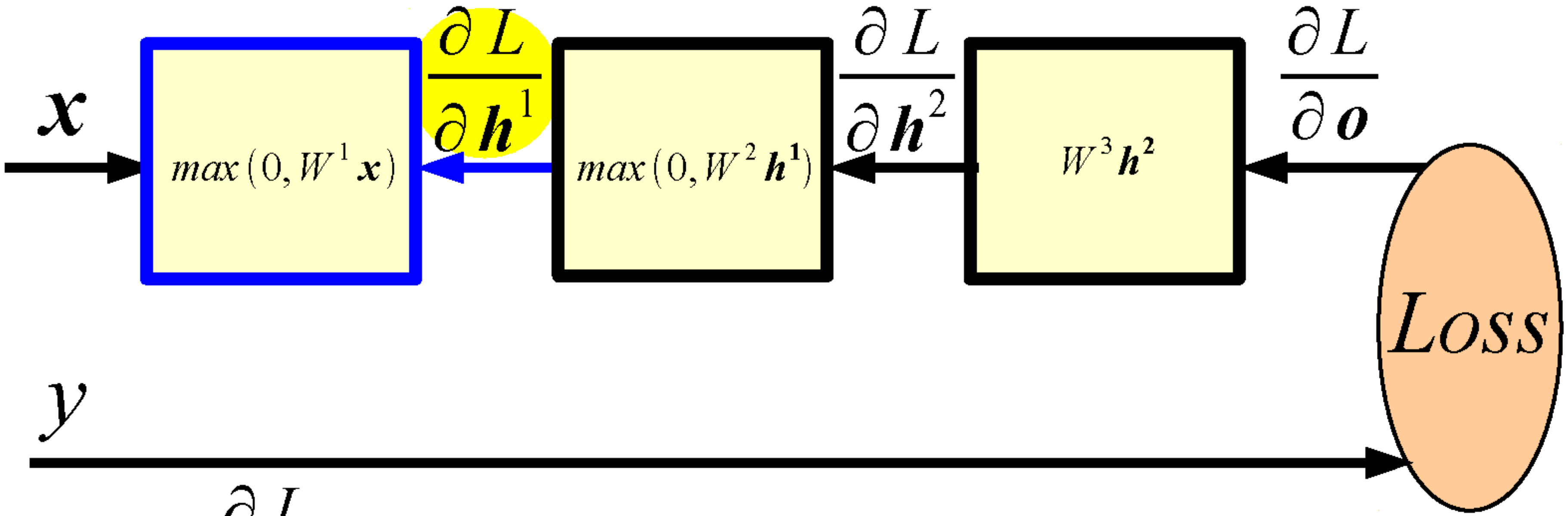
Backward Propagation



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

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

Backward Propagation



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

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

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).

Optimization

Stochastic Gradient Descent (on mini-batches):

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

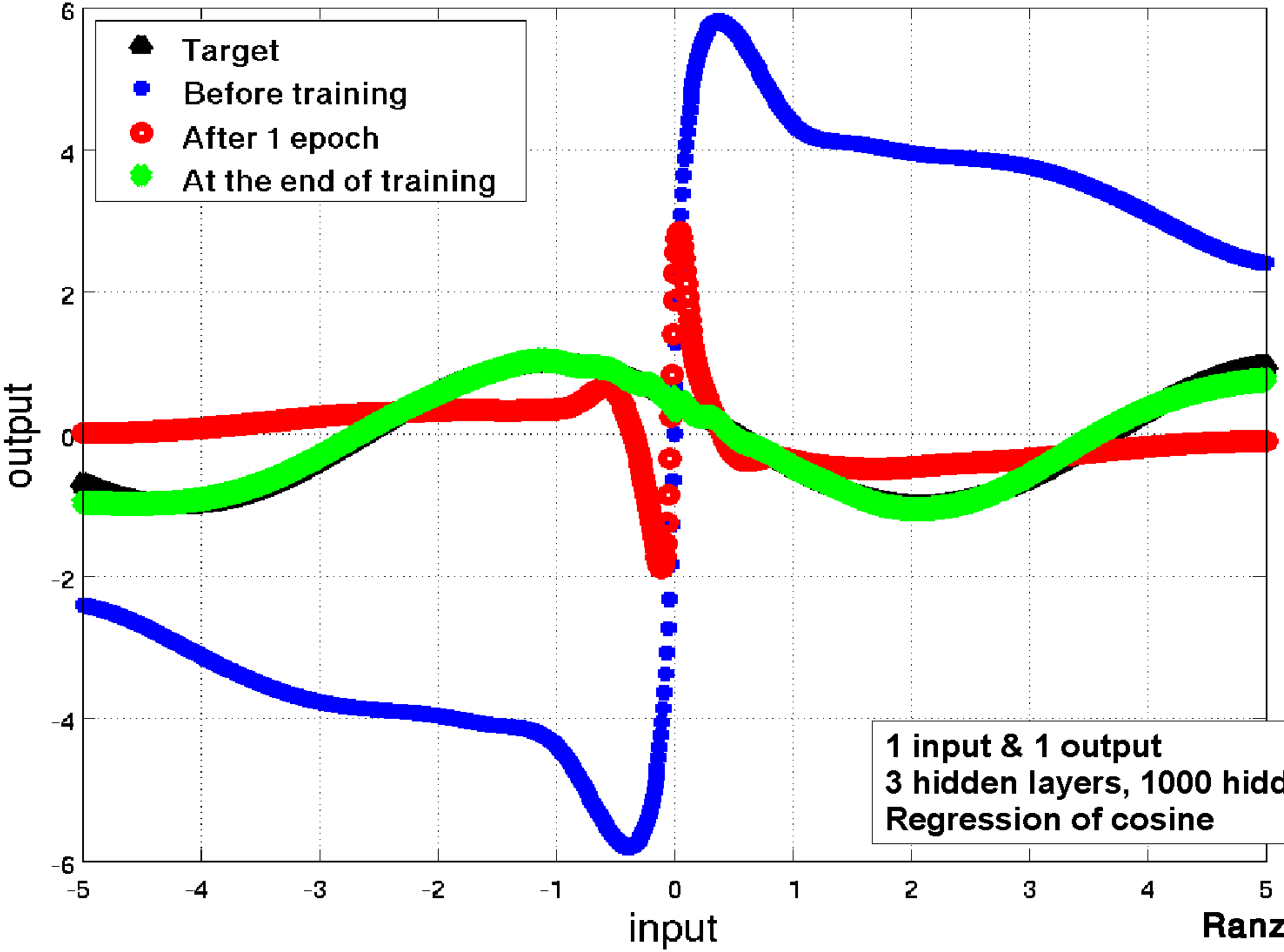
Stochastic Gradient Descent with Momentum:

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

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

Note: there are many other variants...

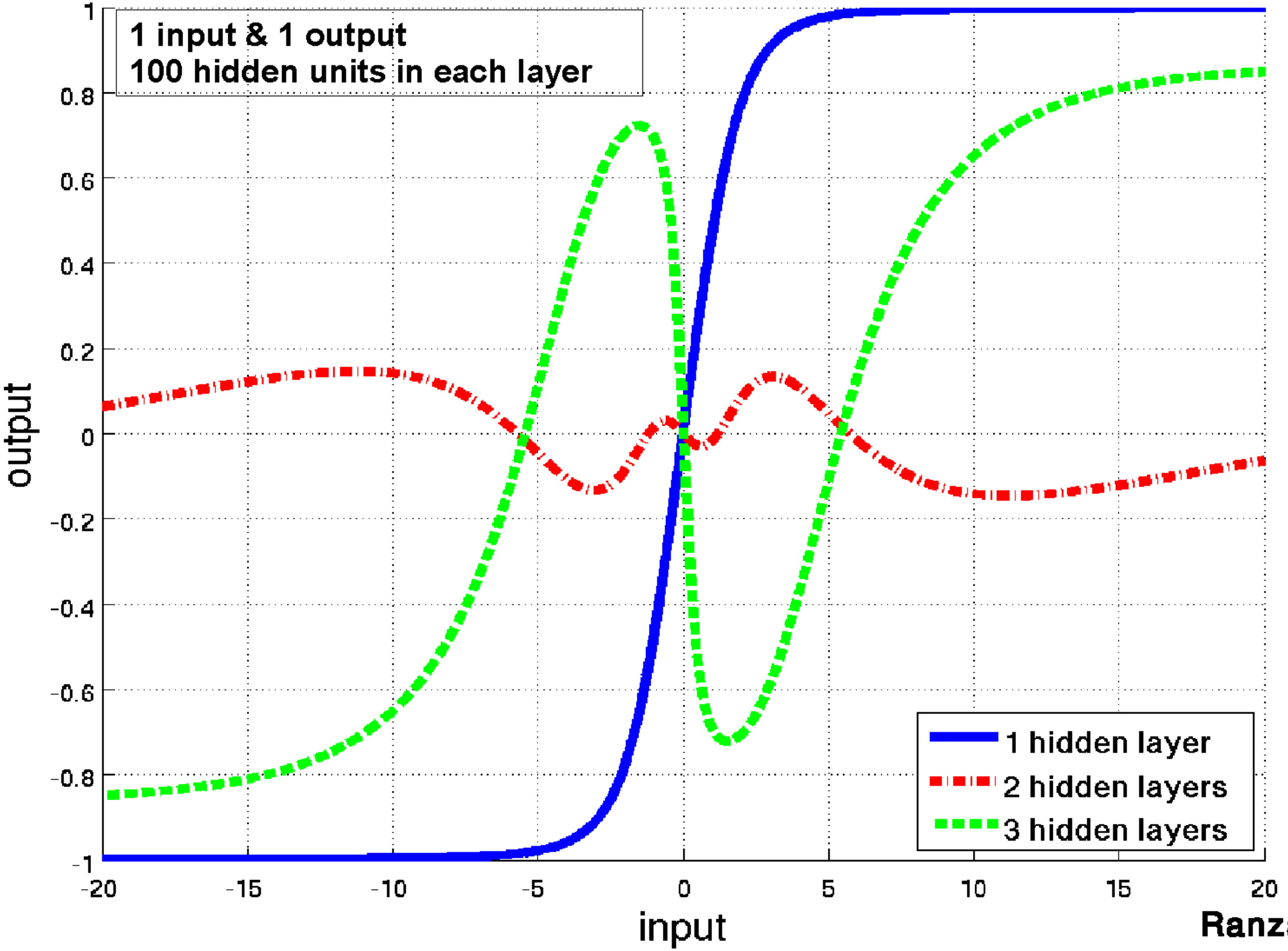
Toy Example: Synthetic Data



1 input & 1 output
3 hidden layers, 1000 hiddens
Regression of cosine

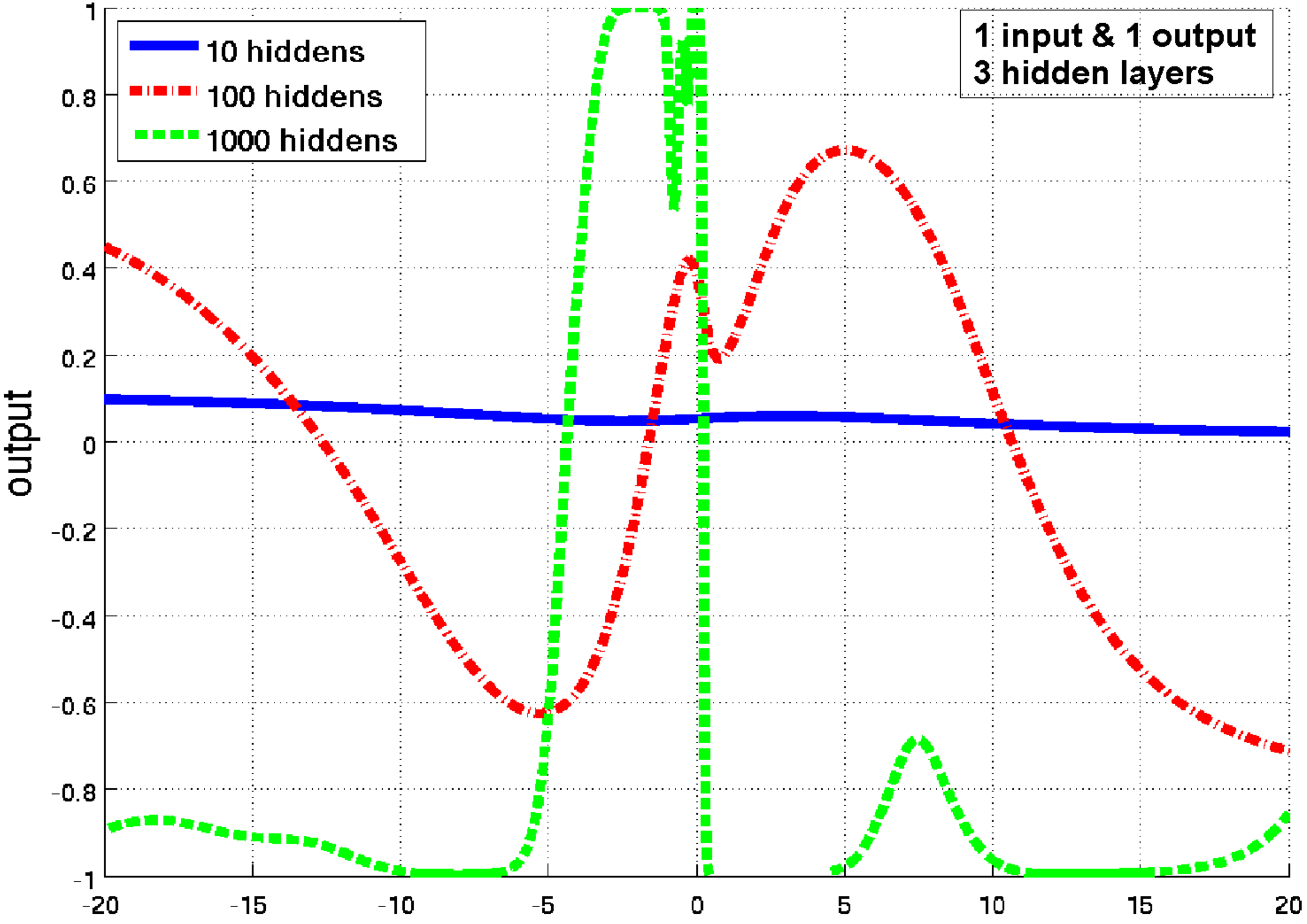
Ranzato

Toy Example: Synthetic Data



Ranzato

Toy Example: Synthetic Data



Ranzato

Outline

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