4^{th} **CiFAR Summer School on**
Learning and Vision in Biology and Engineering
Toronto, August 5-9 2008

Autoencoders, denoising autoencoders, and learning deep networks

Pascal Vincent

Université de Montréal

Laboratoire
d'**I**nformatique
des **S**ystèmes
d'**A**pprentissage
www.iro.umontreal.ca/~lisa

Overview

Part I: Background on Multi-Layer Perceptrons (MLP) and classical autoencoders

- Start from classical models in statistics,
  introduce neural network terminology along the way
  linear regression ➠ ridge regression ➠ logistic regression ➠ multilayer perceptron
- Motivate interest in **deep** multilayer networks
- Introduce classical autoencoder

Part II: Denoising autoencoders for learning deep nets

- Recent work presented at ICML 2008
  P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol
  Extracting and Composing Robust Features with Denoising Autoencoders

Part I

Background on Multi-Layer Perceptrons and classical autoencoders

The (supervised) task

predicting t from x

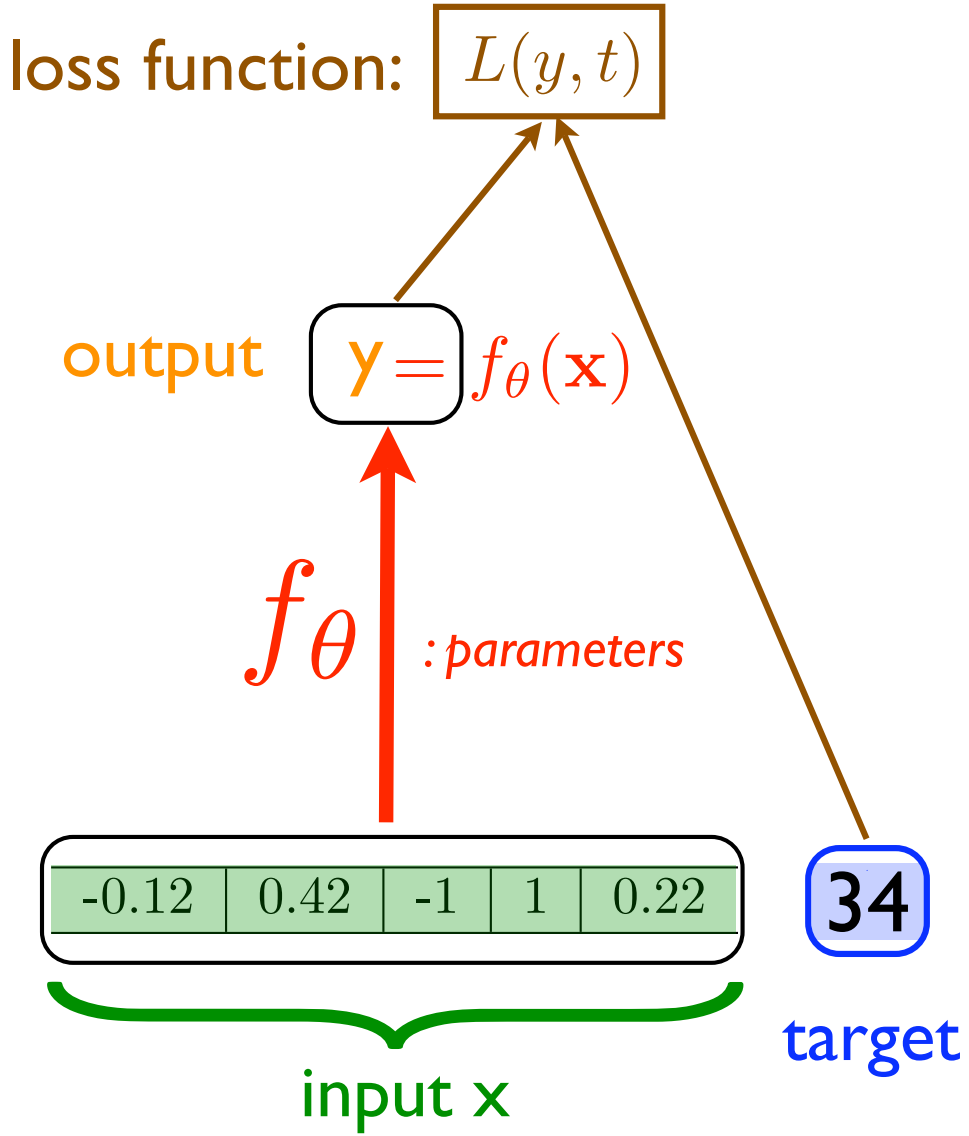
input $\mathbf{x} \in \mathbb{R}^d$ target t

n examples

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	t
0.32	-0.27	+1	0	0.82	113
-0.12	0.42	-1	1	0.22	34
0.06	0.35	-1	1	-0.37	56
0.91	-0.72	+1	0	-0.63	77
...	...	...	...	...	...

Training Set D_n

Learning a parameterized function f_θ that minimizes a loss.



Empirical risk minimization

We need to specify:

- A form for parameterized function f_θ
- A specific loss function $L(y, t)$

We then define the empirical risk as:

$$\hat{R}(f_\theta, D_n) = \sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})$$

i.e. overall loss over the training set

Learning amounts to finding optimal parameters:

$$\theta^\star = \arg\min_\theta \hat{R}(f_\theta, D_n)$$

Linear Regression

A simple learning algorithm

We choose

A **linear** mapping:

$$f_\theta(\mathbf{x}) = \underbrace{\langle \mathbf{w}, \mathbf{x} \rangle}_{\text{dot product}} + b \quad \text{with parameters: } \theta = \{\mathbf{w}, b\}, \underbrace{\mathbf{w}}_{\text{weight vector}} \in \mathbf{R}^d, \underbrace{b}_{\text{bias}} \in \mathbf{R}$$

Squared error loss:

$$L(y, t) = (y - t)^2$$

We search the parameters that minimize the overall loss over the training set

$$\theta^\star = \arg\min_\theta \hat{R}(f_\theta, D_n)$$

Simple linear algebra yields an analytical solution.

Linear Regression

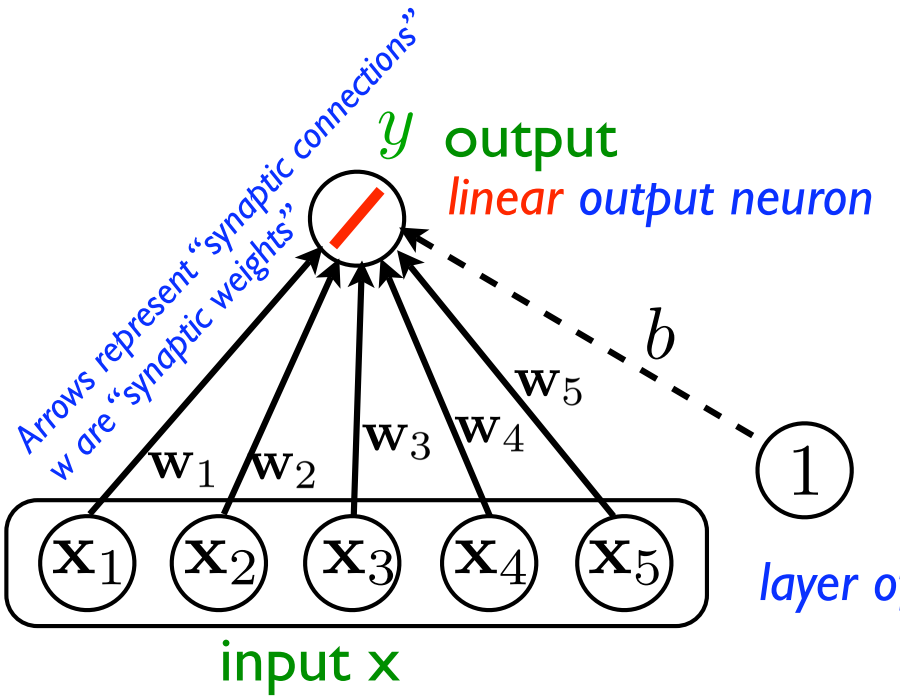
Neural network view

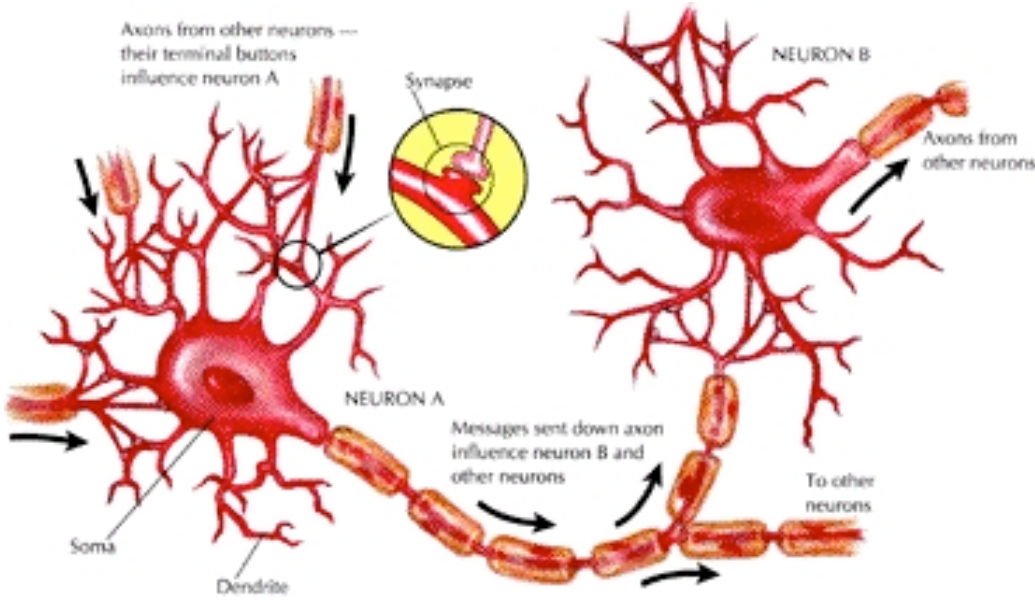
Inuitive understanding of the dot product:

each component of x weighs differently on the response.

$$y = f_\theta(\mathbf{x}) = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \ldots + \mathbf{w}_d\mathbf{x}_d + b$$

Neural network terminology:





Regularized empirical risk

It may be necessary to induce a preference for some values of the parameters over others to avoid "overfitting"

We can define the regularized empirical risk as:

$$\hat{R}_\lambda(f_\theta, D_n) = \underbrace{\left(\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})\right)}_{\text{empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\text{regularization term}}$$

Ω penalizes more or less certain parameter values
$\lambda \geq 0$ controls the amount of regularization

Ridge Regression

= Linear regression + L2 regularization

We penalize large weights:

$$\Omega(\theta) = \Omega(\mathbf{w}, b) = \|\mathbf{w}\|^2 = \sum_{j=1}^{d} \mathbf{w}_j^2$$

In neural network terminology:

"weight decay" penalty

Again, simple linear algebra yields an analytical solution.

Logistic Regression

If we have a binary classification task: $t \in \{0, 1\}$

We want to estimate conditional probability: $y \simeq P(t = 1|\mathbf{x})$

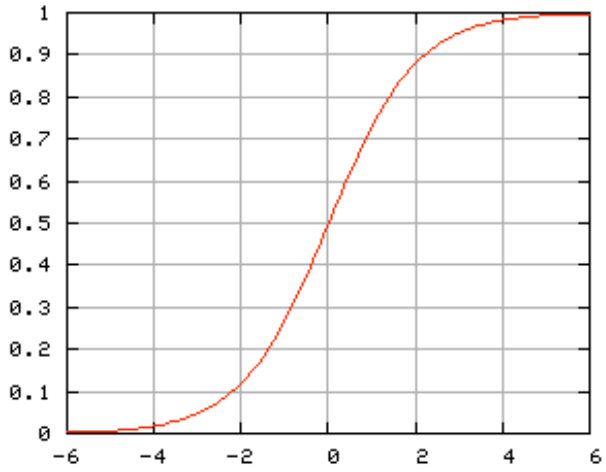
$y \in [0, 1]$

We choose

A **non-linear** mapping:

$$f_\theta(\mathbf{x}) = f_{\mathbf{w},b}(\mathbf{x}) = \underbrace{\text{sigmoid}(}_{\text{non-linearity}} \langle \mathbf{w}, \mathbf{x} \rangle + b)$$

logistic $$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



The *logistic sigmoid* is the inverse of the *logit* "link function" in the terminology of Geleralized Linear Models (GLMs).

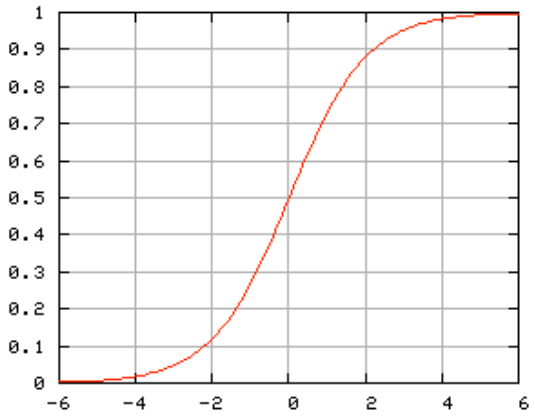
Cross-entropy loss:

$$L(y, t) = t \ln(y) + (1 - t) \ln(1 - y)$$

No analytical solution,
but optimization is convex

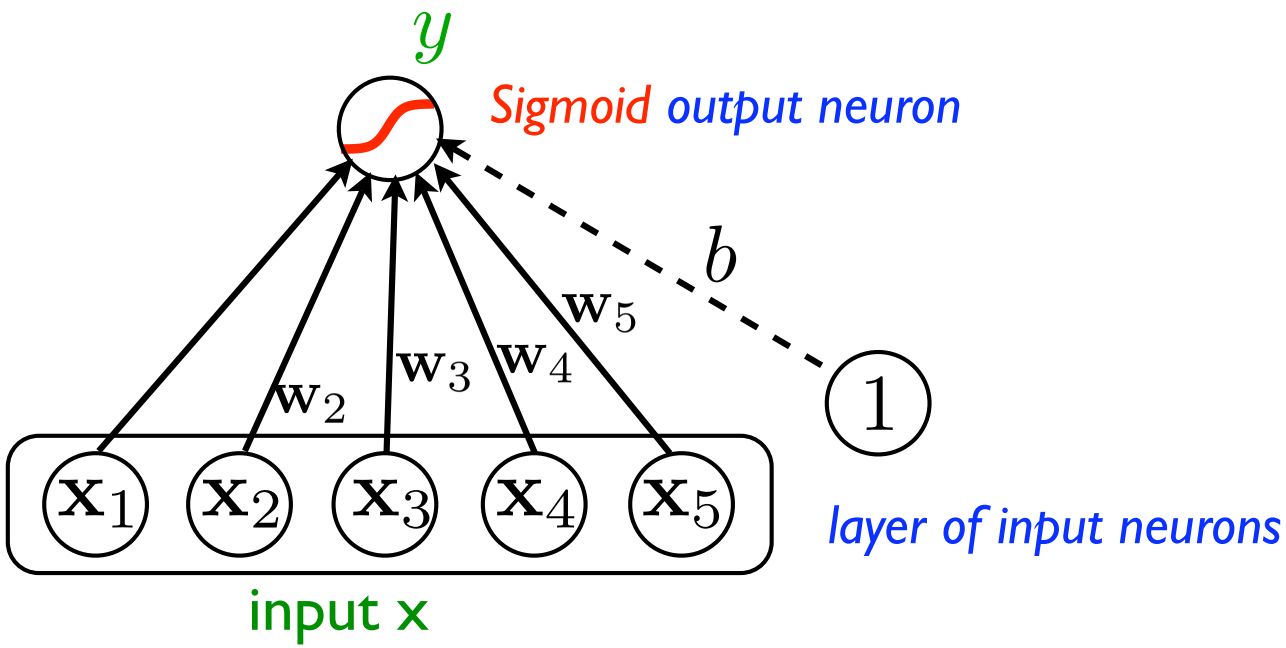
Logistic Regression

Neural network view



Sigmoid can be viewed as:

- "soft" differentiable alternative to the step function of original Perceptron (Rosenblatt 1957).
- simplified model of "firing rate" response in biological neurons.



Only yields "linear" decision boundary: a hyperplane

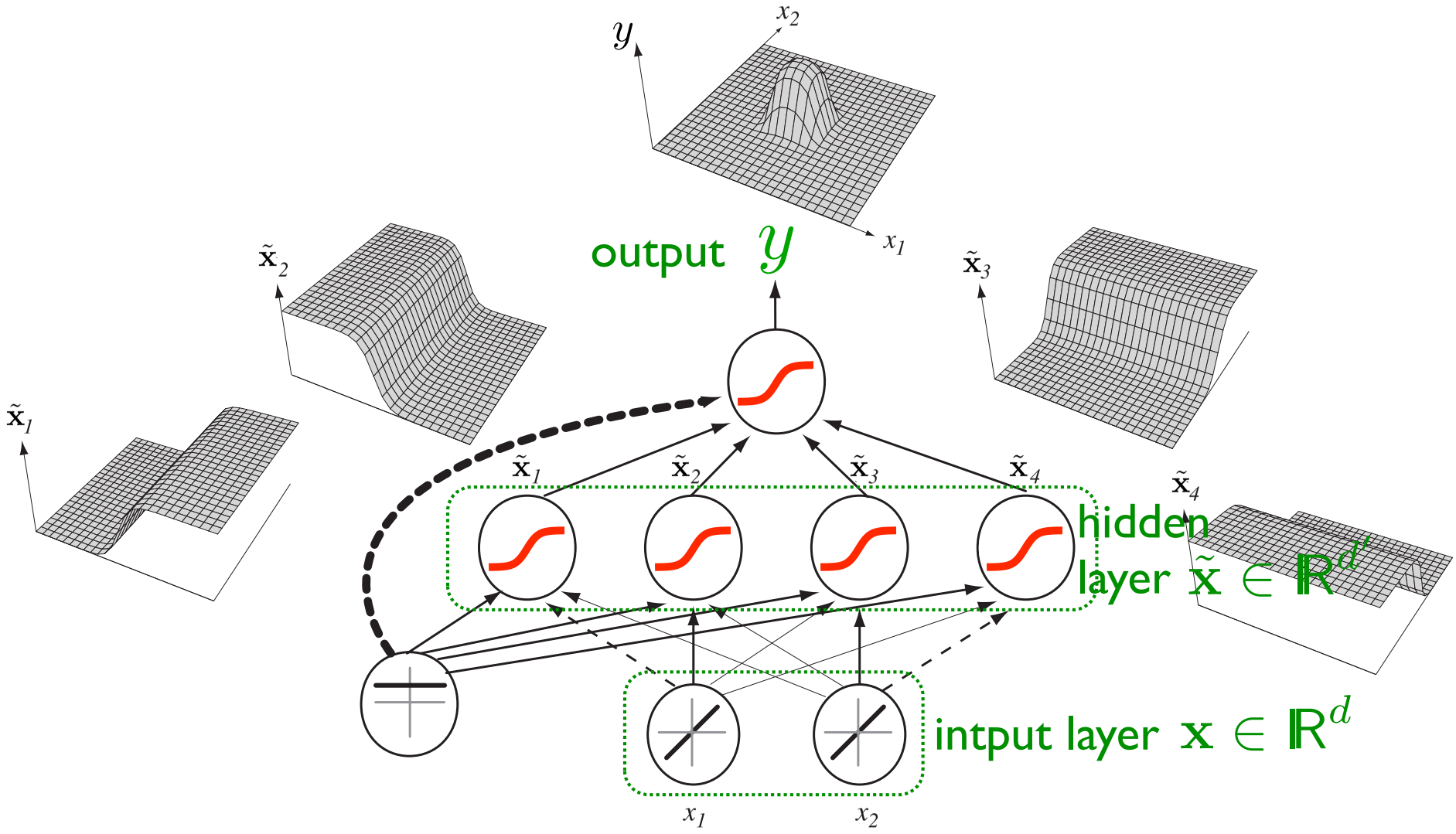
➠ inappropriate if classes not linearly separable

How to obtain non-linear decision boundaries...

Three ways to map x non-linearly to $\tilde{\mathbf{x}} = \phi(\mathbf{x})$

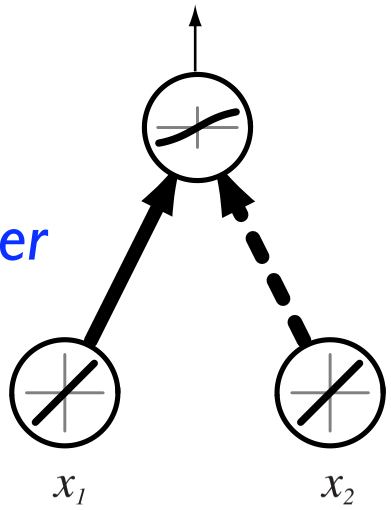
- Use an explicit fixed mapping
  ➠old technique (e.g. using hand-crafted "features")
- Use an implicit fixed mapping
  ➠Kernel Methods (SVMs, Kernel Logistic Regression ...)
- Learn a parameterized mapping:
  ➠ Multi-layer feed-forward Neural Networks
  such as Multilayer Perceptrons (MLP)

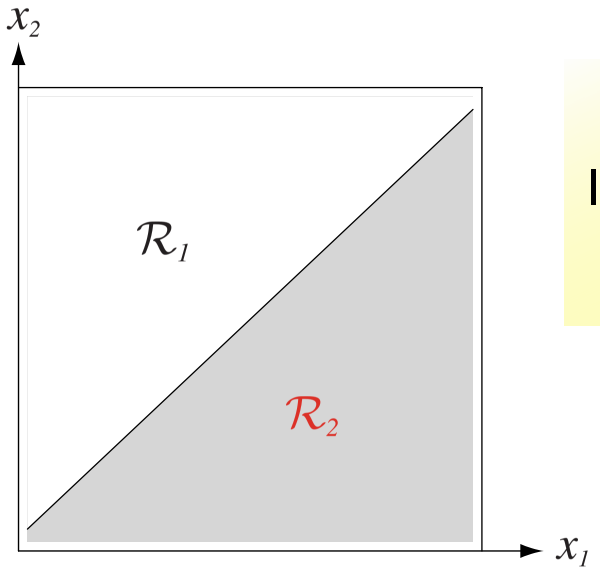
Multi-Layer Perceptron (MLP)
with one hidden layer of size 4 neurons



Expressive power of MLP with one hidden layer

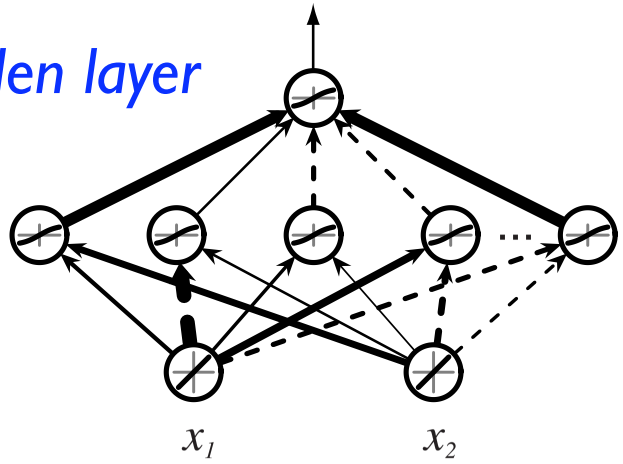
no hidden layer

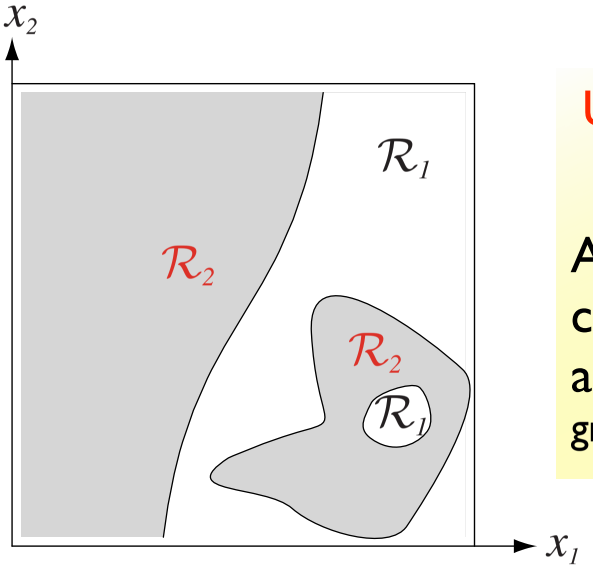




== Logistic regression limited to representing a separating hyperplane

one hidden layer





Universal approximation property

Any continuous function can be approximated arbitarily well (with a growing number of hidden unis)

Multi-Layer Perceptron (MLP)

with one hidden layer of size *d'* neurons

Functional form (parametric):

$$y = f_\theta(\mathbf{x}) = \text{sigmoid}\left(\langle \mathbf{w}, \underbrace{\tilde{\mathbf{x}}}\rangle + b\right)$$

$$\tilde{\mathbf{x}} = \text{sigmoid}(\underbrace{\mathbf{W}^{\text{hidden}}}_{d' \times d}\mathbf{x} + \underbrace{\mathbf{b}^{\text{hidden}}}_{d' \times 1})$$

Parameters:

$$\theta = \{\mathbf{W}^{\text{hidden}}, \mathbf{b}^{\text{hidden}}, \mathbf{w}, b\}$$

Optimizing parameters on training set *(training the network)*:

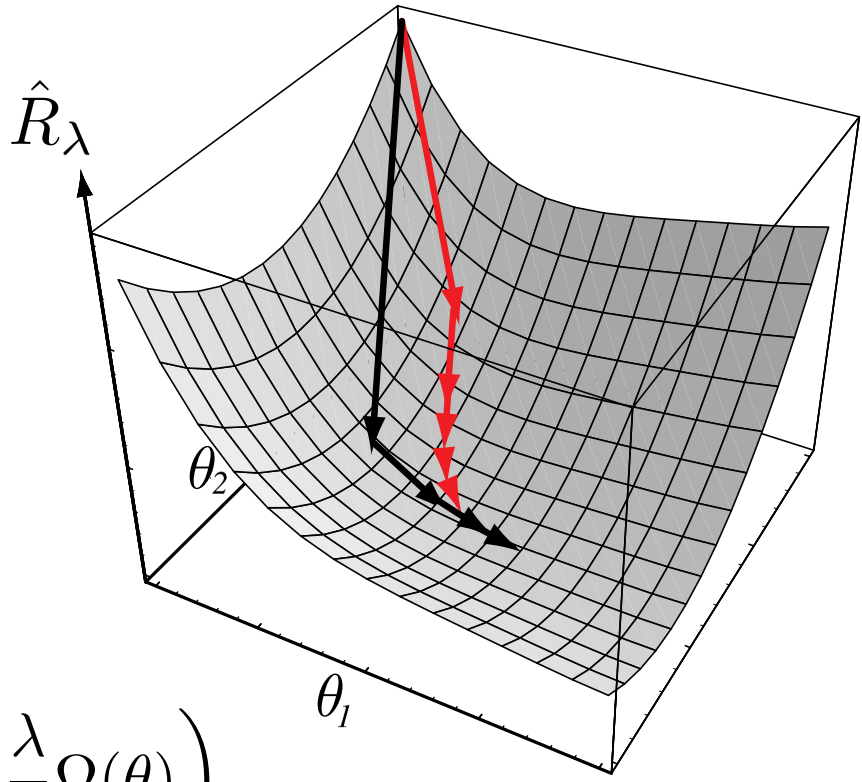
$$\theta^\star = \arg\min_\theta \underbrace{\hat{R}_\lambda(f_\theta, D_n)}_{\left(\underbrace{\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})}_{\text{empirical risk}}\right) + \underbrace{\lambda\Omega(\theta)}_{\substack{\text{regularization term} \\ \text{(weight decay)}}}}$$

Training feed-forward nets

We need to optimize the network's parameters:

$$\theta^\star \quad = \quad \arg\min_\theta \hat{R}_\lambda(f_\theta, D_n)$$

- Initialize parameters (at random...)
- Perform gradient descent



Either batch gradient descent:

REPEAT: $\theta \leftarrow \theta - \eta \frac{\partial \hat{R}_\lambda}{\partial \theta}$

Or stochastic gradient descent:

REPEAT:

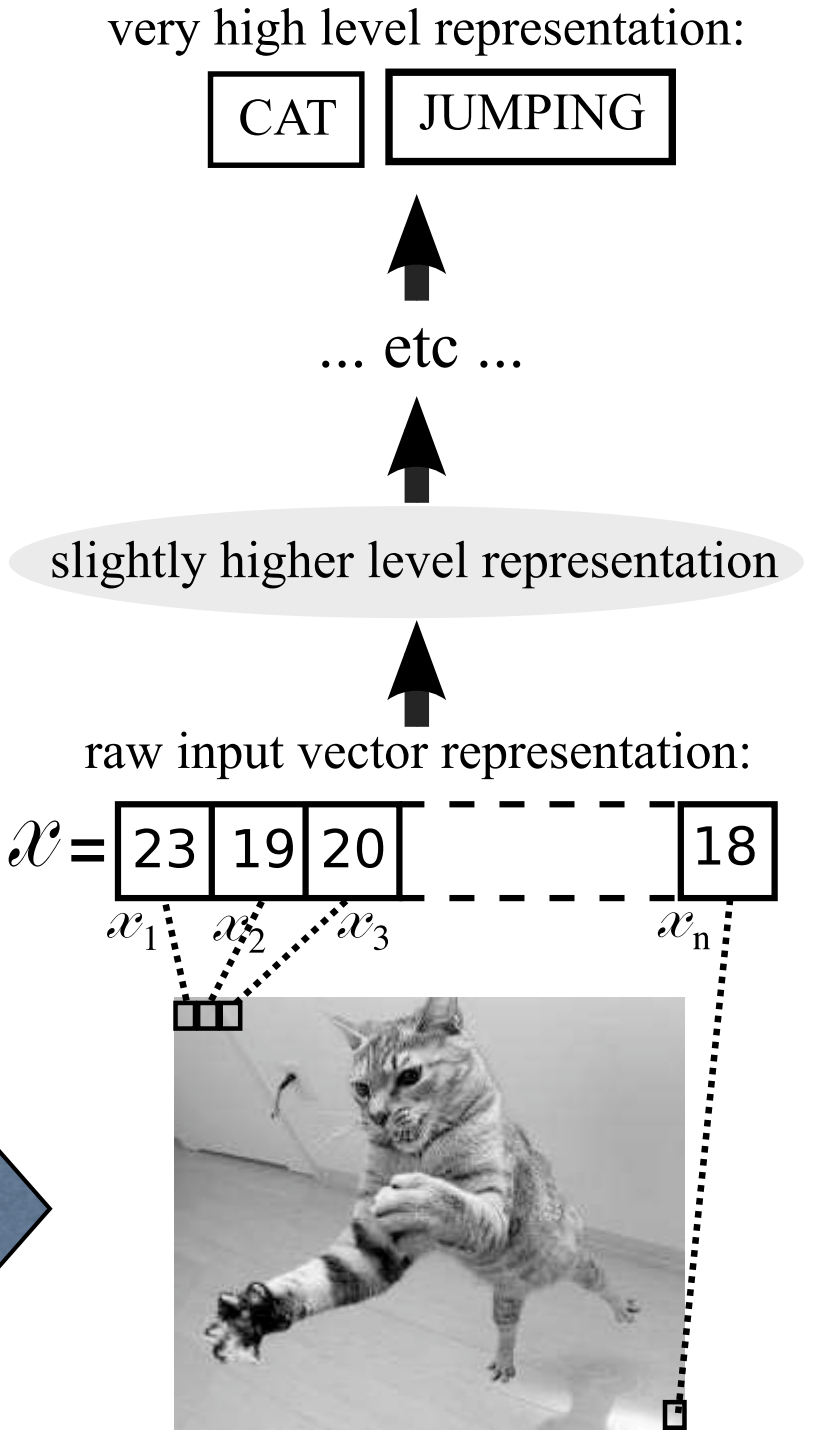
Pick i in 1...n

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \left(L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) + \frac{\lambda}{n} \Omega(\theta) \right)$$

Or other gradient descent technique

(conjugate gradient, Newton, steps natural gradient, ...)

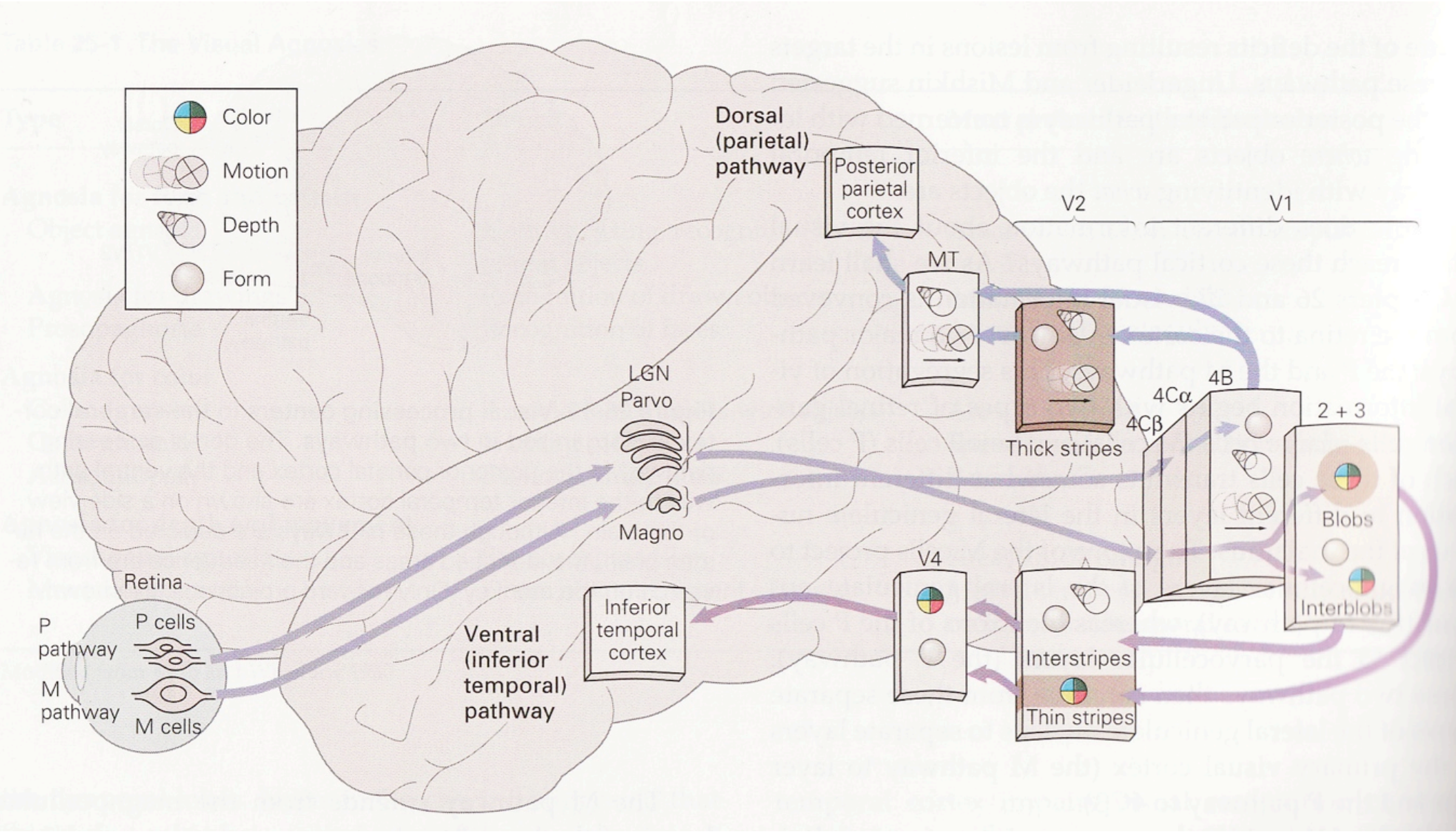
Deep networks:
Multiple layers
as levels of representation



Why the interest in learning deep multi-layer Neural Nets?

- Representational power of functional composition.
- Shallow architectures
  (NNets with one hidden layer, SVMs, boosting, ...)
  can be universal approximators...
- But may require exponentially more nodes than corresponding deep architectures (see Bengio 2007).
- ➠ statistically more efficient to learn small deep architectures (fewer parameters) than fat shallow architectures.

Example of a deep architecture made of multiple layers, solving complex problems...



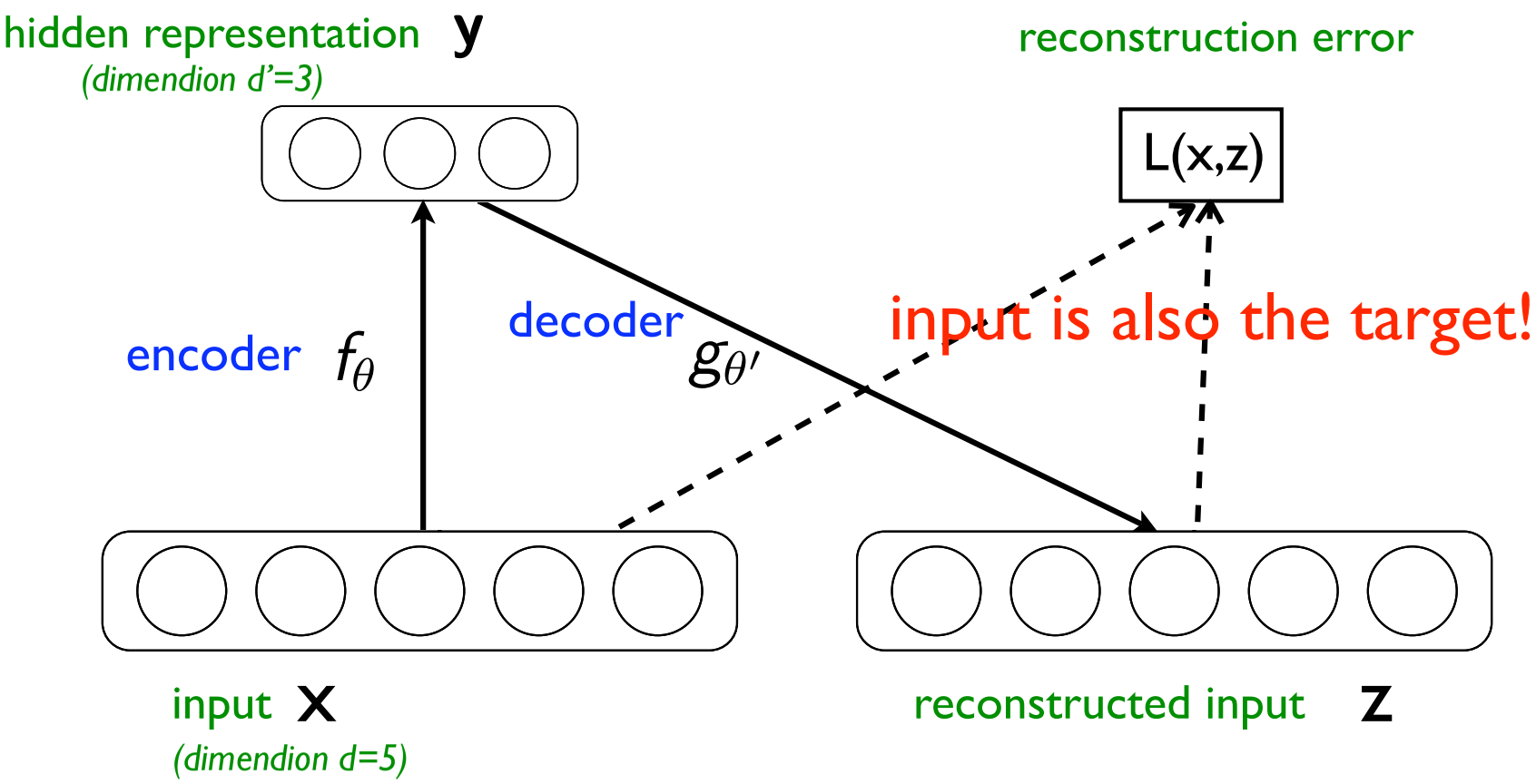
The classical auto-encoder framework

Unsupervised learning: no explicit target t

Goal: model distribution or capture underlying structure in x

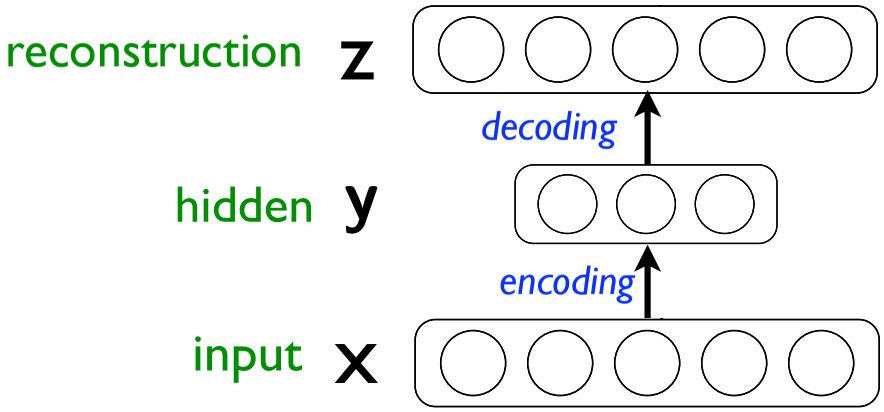
classical auto-encoder learns lower-dimensional representation

➠ dimensionality reduction



Auto-encoders are also called...

- Autoencoders
- Auto-associators
- Diabolo networks



lower-dimensional
bottleneck

The Diabolo

Classical auto-encoder facts

- With linear neurons and squared loss
  ➠ autoencoder learns same suspace as PCA
- Also true for a single non-linear hidden layer, if using linear output neurons with squared loss
- We will be using sigmoidal neurons and cross-entropy loss.
- One can use more than one hidden layers