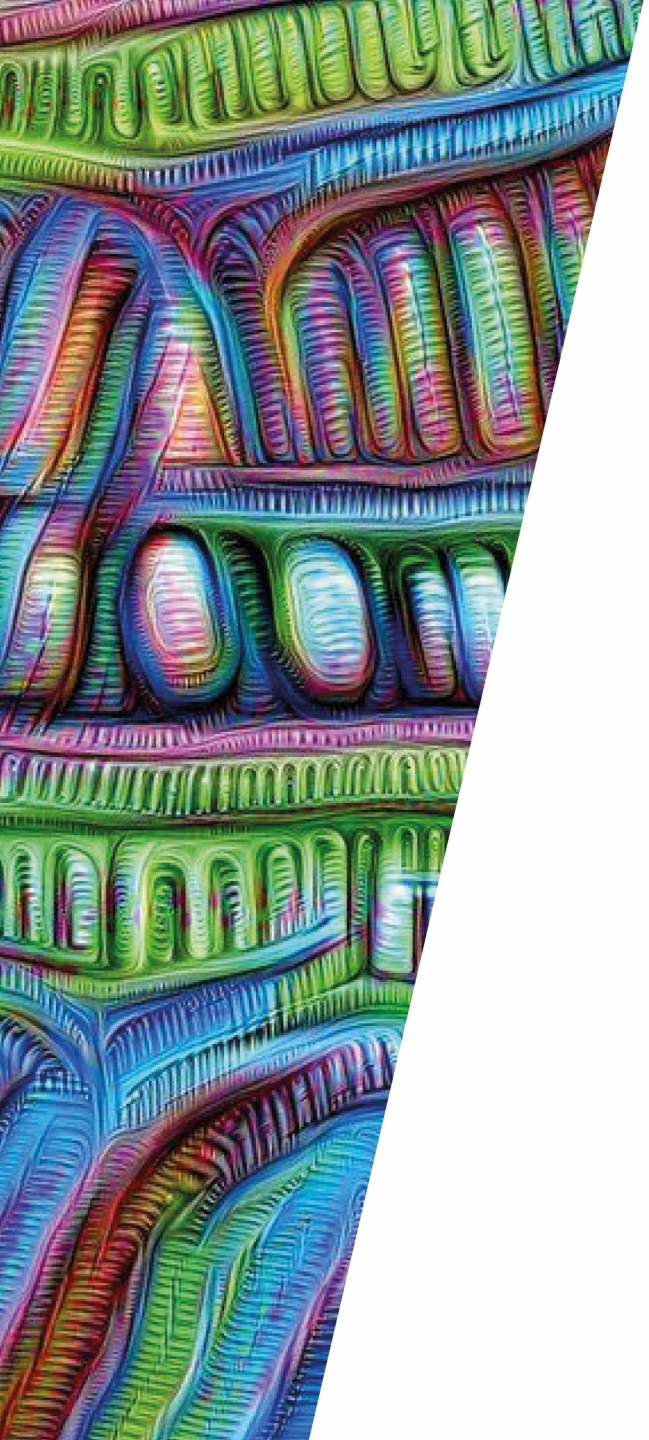


Bienvenue!

ÉCOLE D'ÉTÉ FRANCOPHONE EN APPRENTISSAGE PROFOND

21-25 août 2017

IVADO

HEC Montréal
Polytechnique Montréal
Université de Montréal

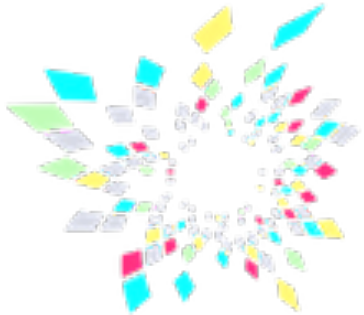
MILA

Réseaux de neurones multi-couches

Yoshua Bengio

22 août 2017

École d'Été en Apprentissage Profond



Couche d'entrée, couches cachées, couche de sortie

Output layer

Here predicting a supervised target

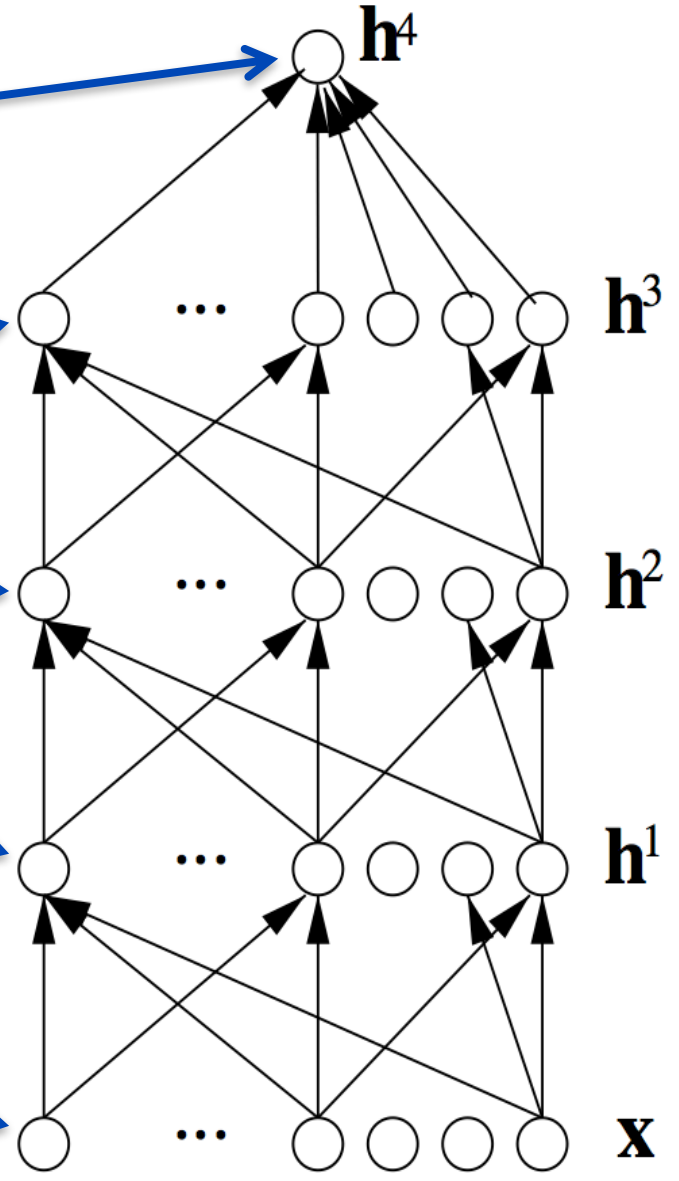
Hidden layers

These learn more abstract representations as you head up

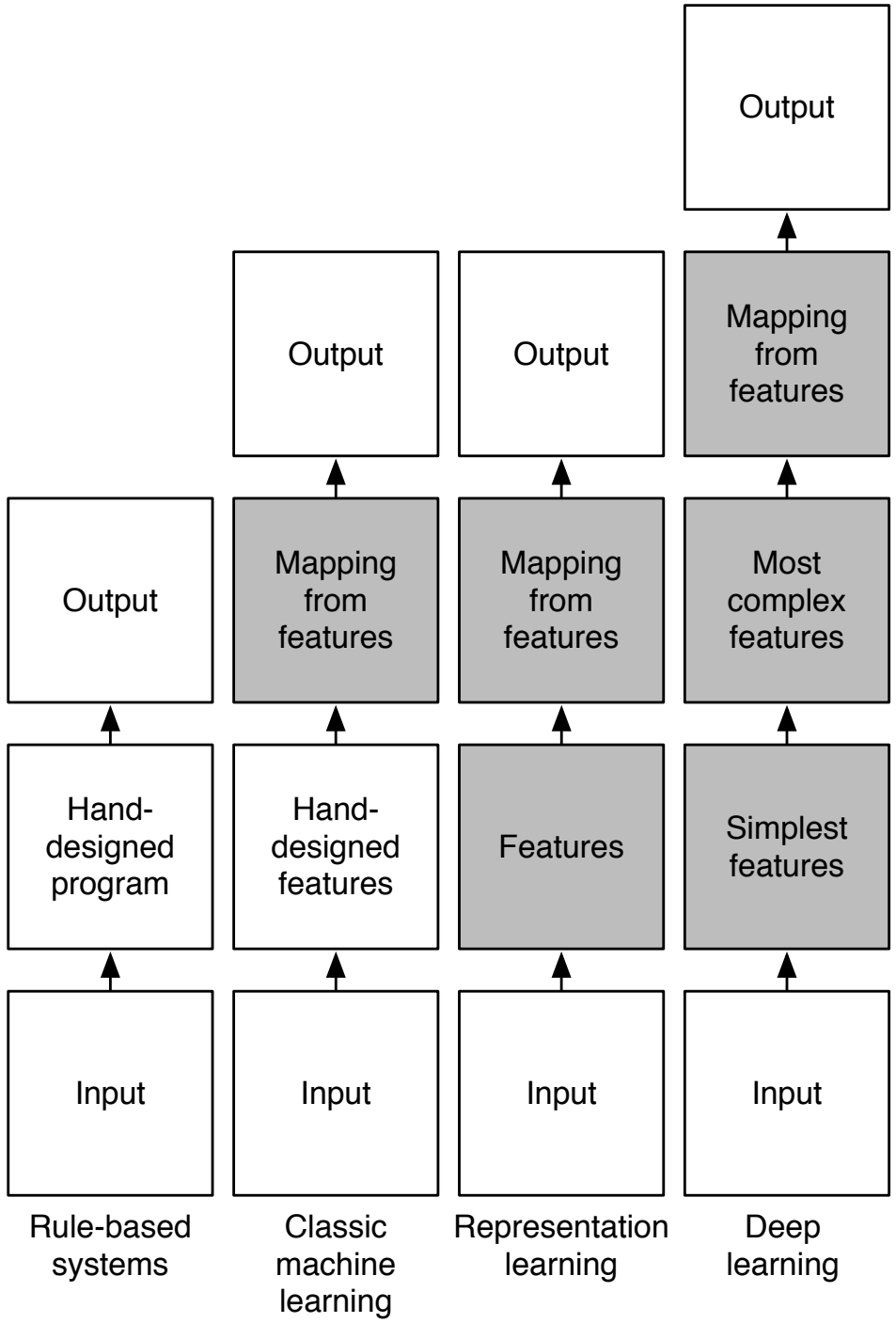
Input layer

This has raw sensory inputs

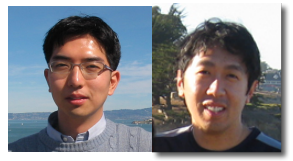
(roughly)



Apprentissage de plusieurs transformations successives, de caractéristiques, de représentations

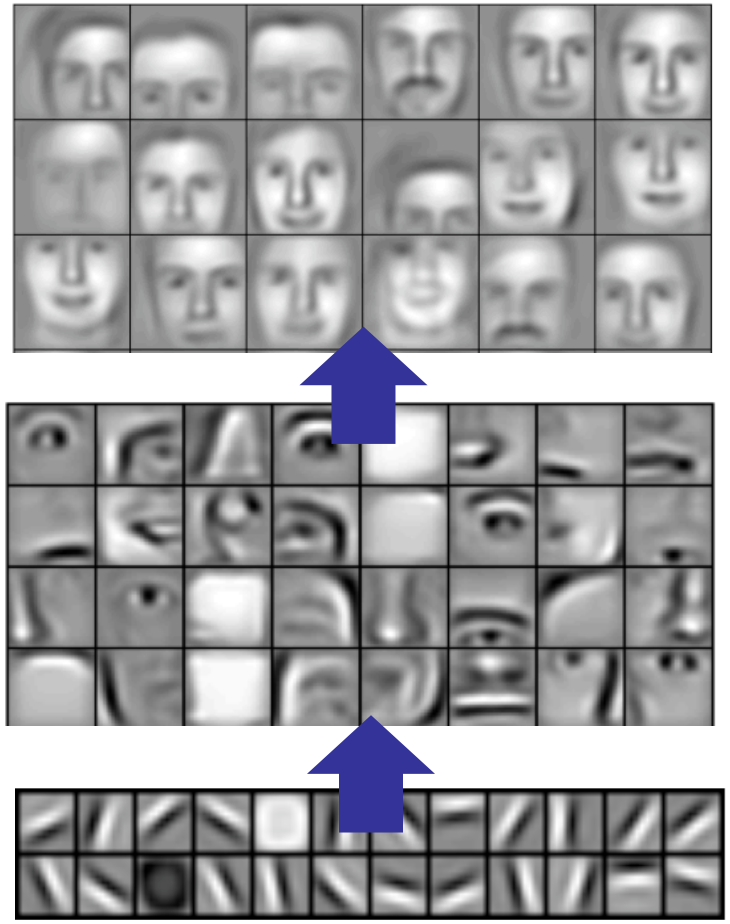


Learning multiple levels of representation



(Lee, Largman, Pham & Ng, NIPS 2009)
(Lee, Grosse, Ranganath & Ng, ICML 2009)

Successive model layers learn deeper intermediate representations

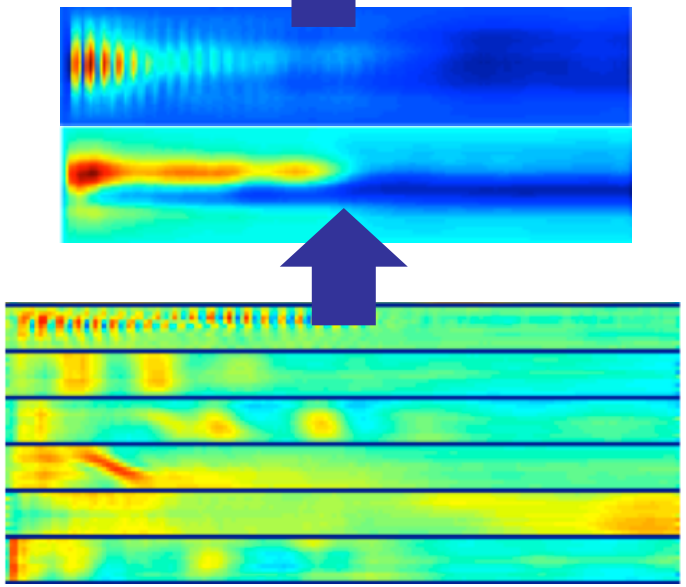


Layer 3

High-level
linguistic representations

Parts combine
to form objects

Layer 2

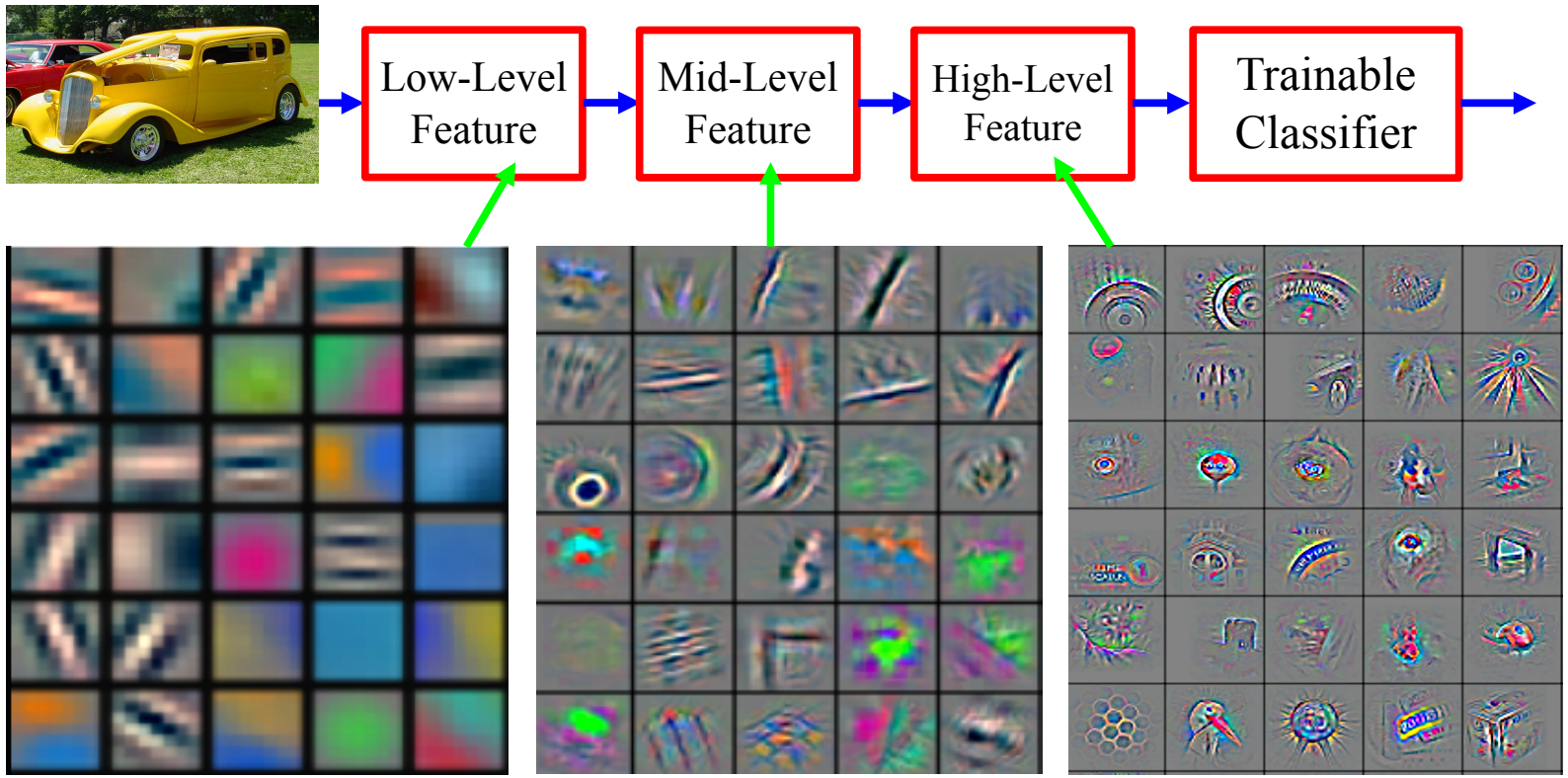


Layer 1

Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction

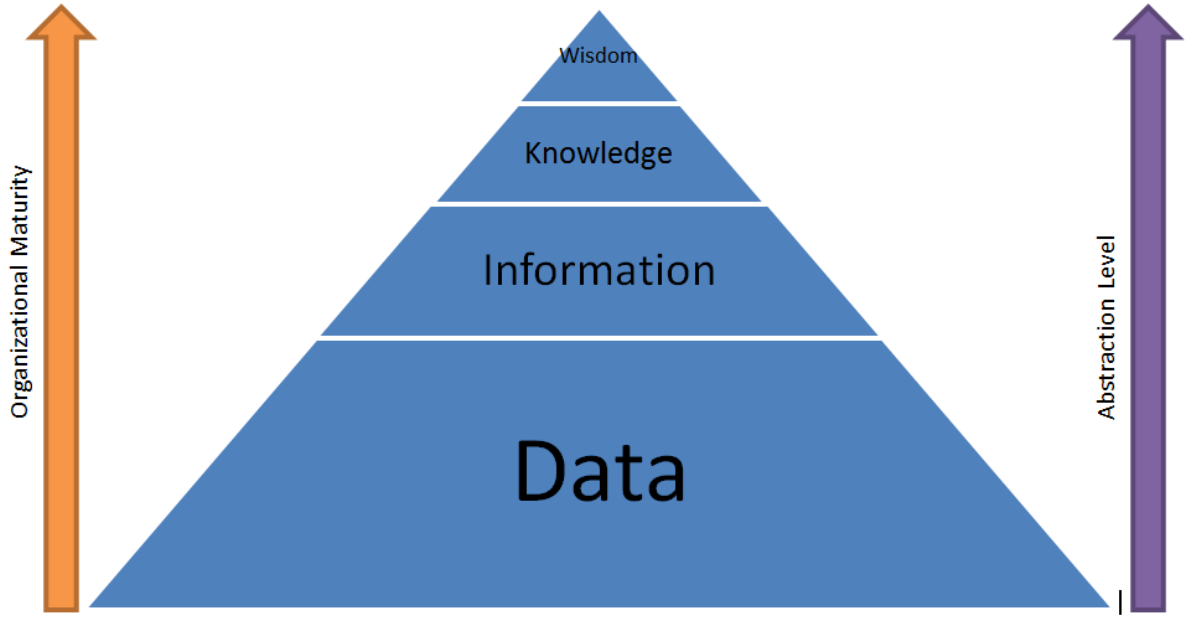
Why Multiple Layers? The World is Compositional

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel → edge → texton → motif → part → object
- Text: Character → word → word group → clause → sentence → story
- Speech: Sample → spectral band → sound → … → phone → phoneme → word



Apprendre plusieurs niveaux d'abstraction

- The big payoff of deep learning is to allow learning higher levels of abstraction
- Higher-level abstractions disentangle the factors of variation, which allows much easier generalization and transfer



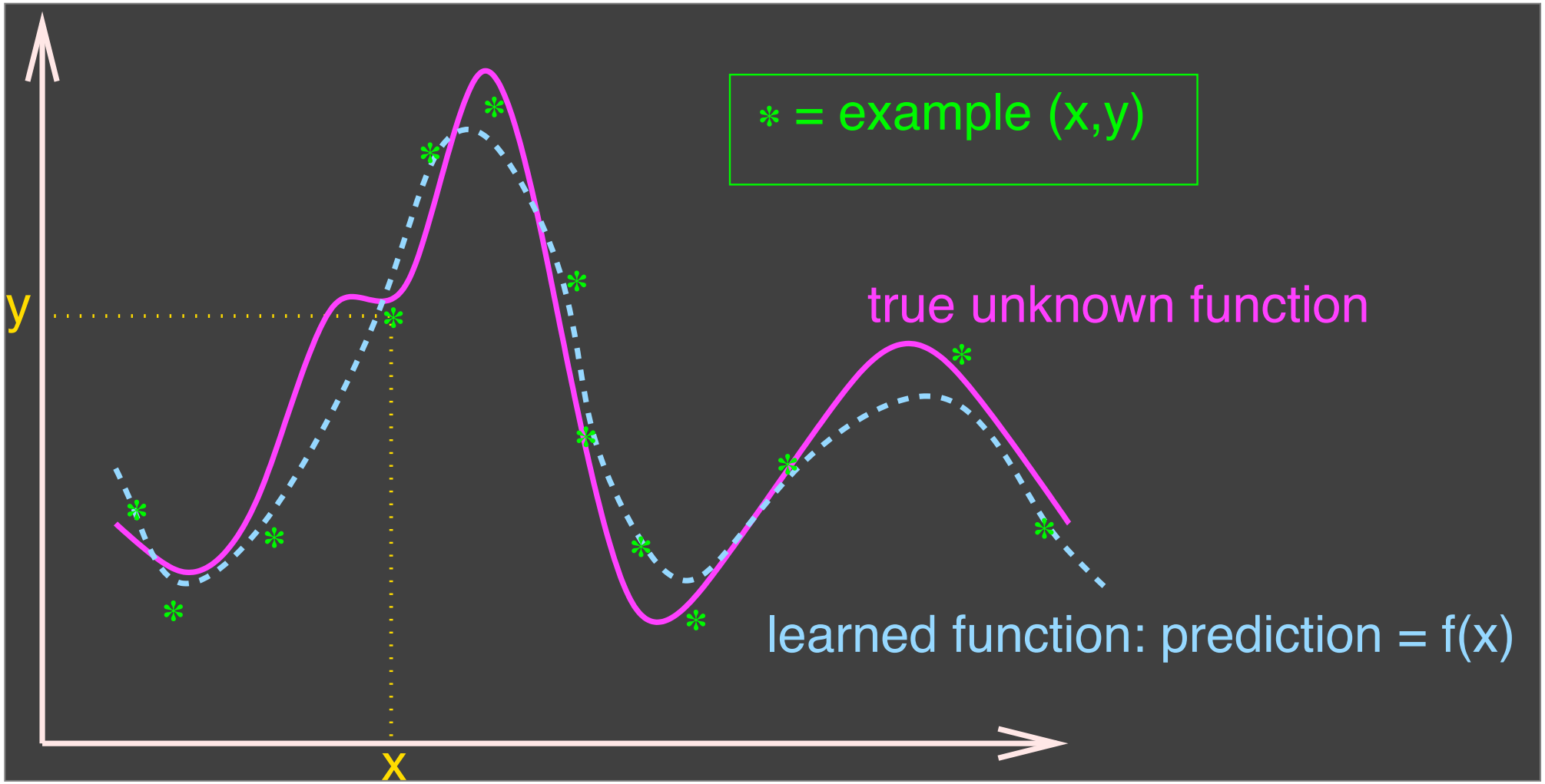
Apprentissage profond: basé sur une représentation interne apprise

- Diffère des méthodes d'apprentissage automatique pour lesquelles il n'y a pas de représentation interne (e.g. régression linéaire ou logistique) ou bien où la représentation interne n'est pas apprise (caractéristiques définies par un expert, ou bien méthodes à noyau, SVMs, etc.)

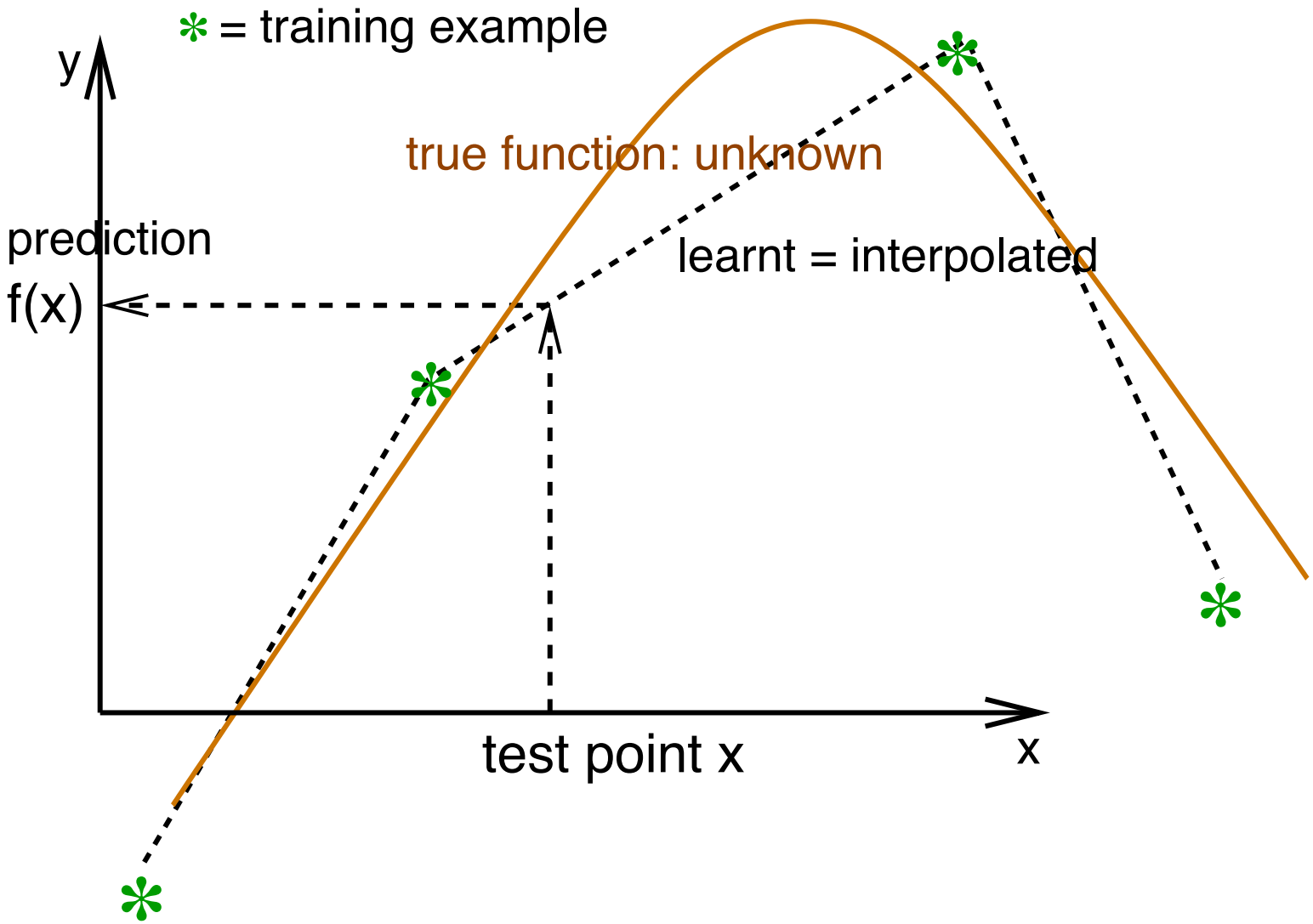
- La représentation interne est apprise de façon à rendre d'autres tâches plus faciles (classification, modélisation de la structure de la distribution, etc.): certaines opérations simples y ont un sens

- Si elle n'est pas apprise, elle pourrait devoir être de très haute dimension (SVMs, méthodes à noyau)

Easy Learning



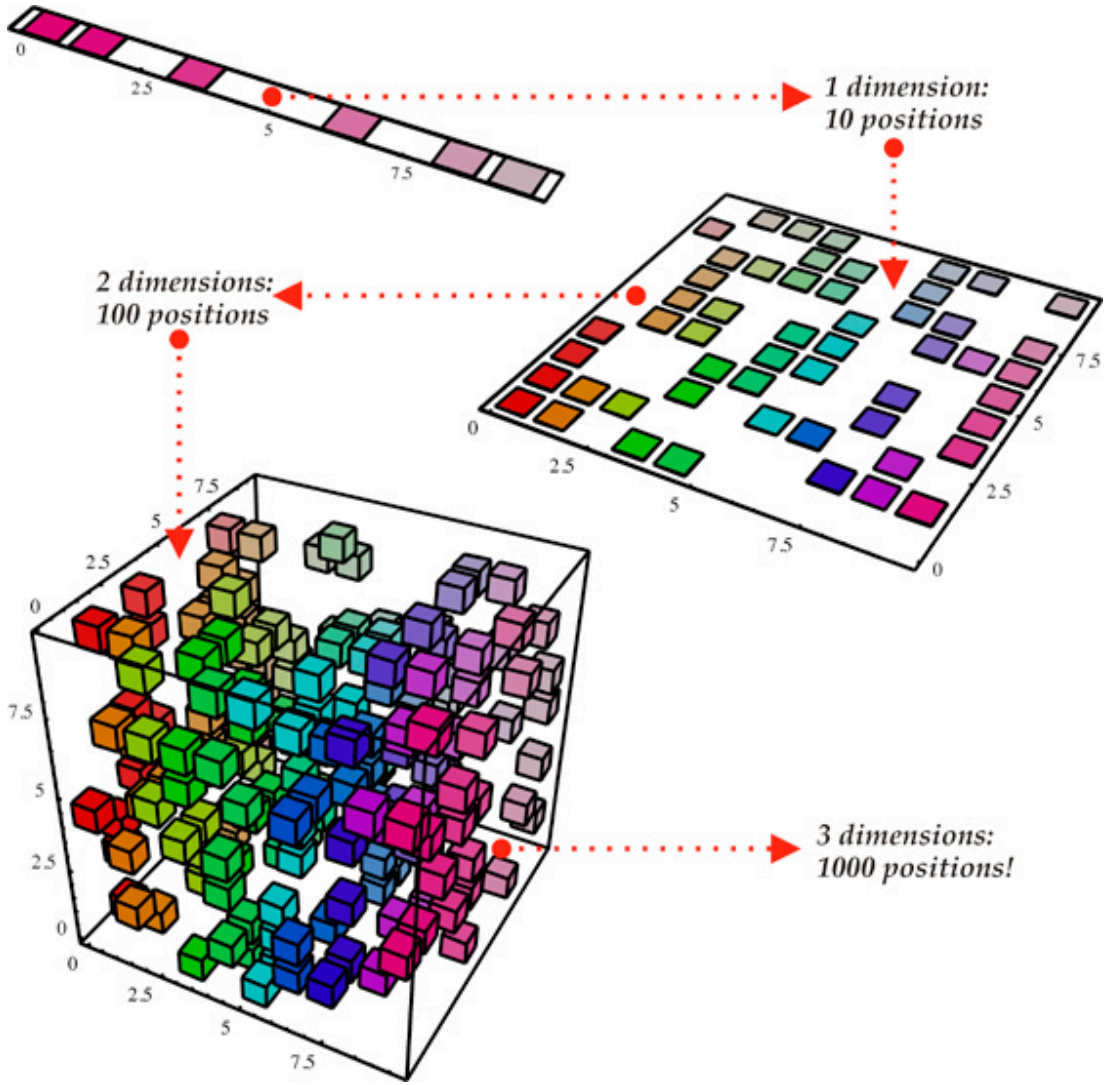
Local Smoothness Prior: Locally Capture the Variations



ML 101. What We Are Fighting Against: The Curse of Dimensionality

To generalize locally, need representative examples for all relevant variations!

Classical solution: hope for a smooth enough target function, or make it smooth by handcrafting good features / kernel



Bypassing the curse of dimensionality

We need to build compositionality into our ML models

- Just as human languages exploit compositionality to give representations and meanings to complex ideas

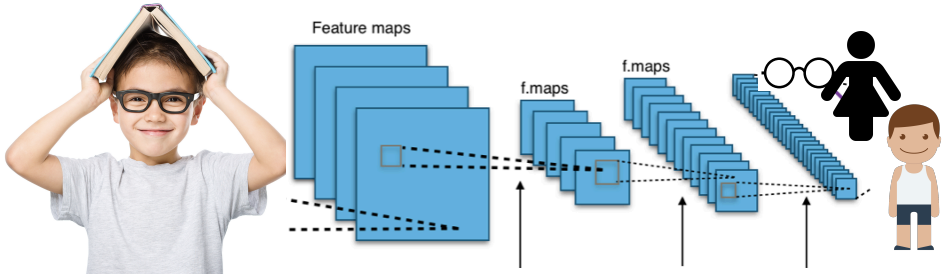
Exploiting compositionality gives an exponential gain in representational power

- Distributed representations / embeddings: feature learning
- Deep architecture: multiple levels of feature learning

Prior: compositionality is useful to describe the world around us efficiently

Each feature can be discovered without the need for seeing the exponentially large number of configurations of the other features

- Consider a network whose hidden units discover the following features:
  - Person wears glasses
  - Person is female
  - Person is a child
  - Etc.



If each of n feature requires k parameters, need on the order of nk examples

Non-parametric methods would require on the order of n^d examples

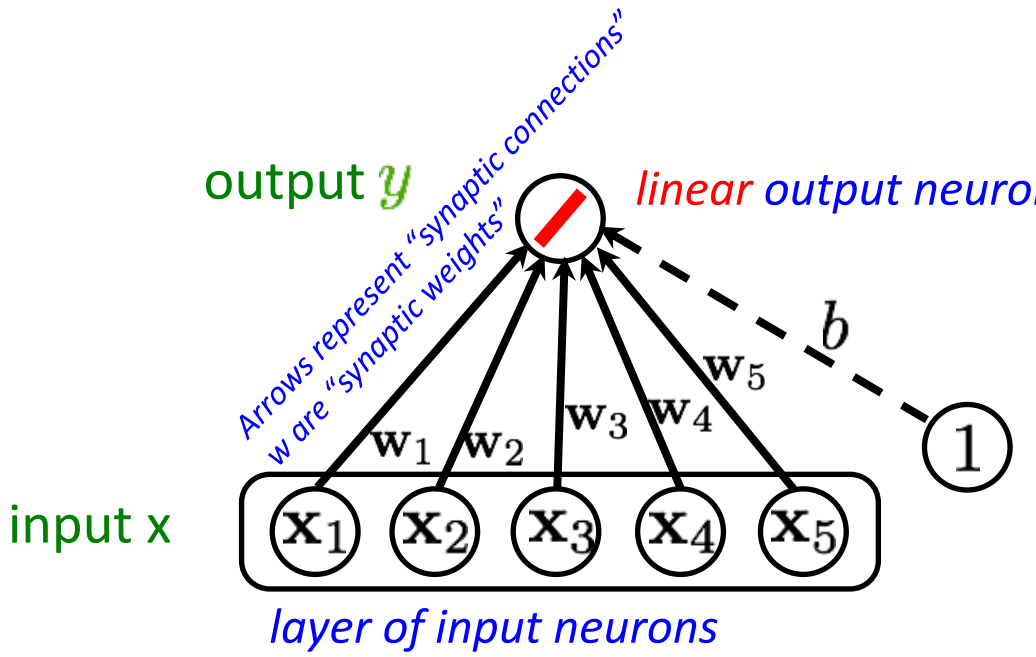
Régression Linéaire, neurone Linéaire

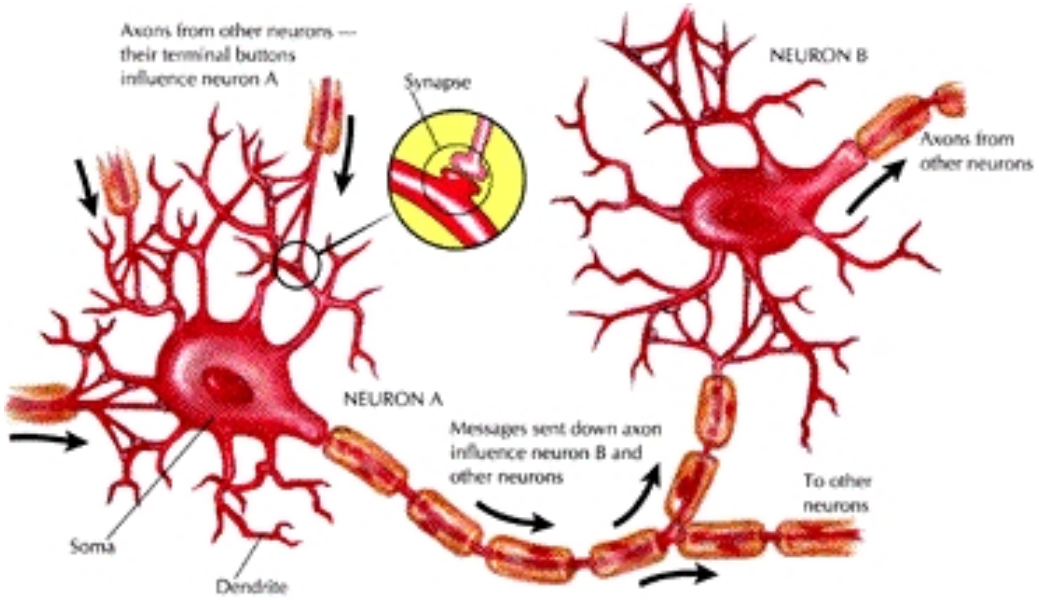
Intuitive understanding of the dot product:

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:





Estimer une espérance conditionnelle avec l'erreur quadratique

Pour prédire une quantité continue, minimiser l'erreur quadratique

- Si le critère de perte $L = (y - f_\theta(x))^2$ est l'erreur quadratique

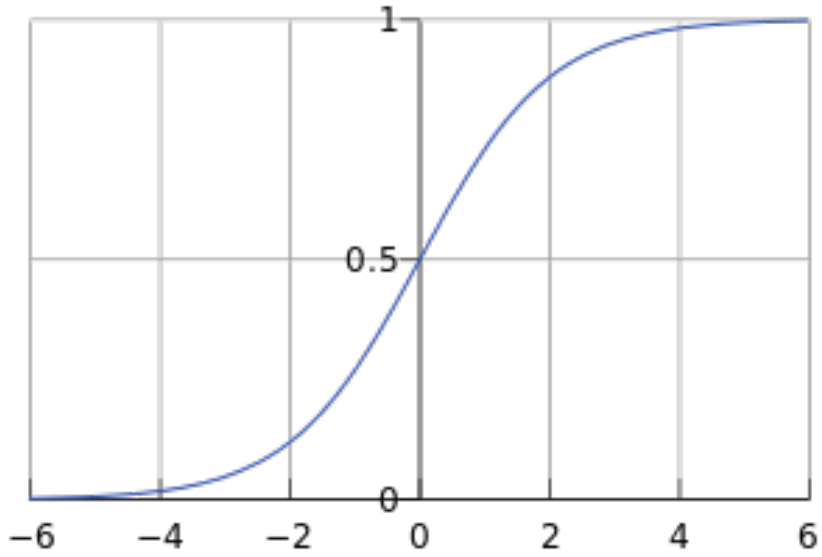
$$min_\theta E[(y - f_\theta(x))^2]$$

- alors le réseau de neurones estime la valeur moyenne des *y* possibles pour le *x* donné:

$$\Rightarrow \quad f_\theta(x) \approx E[y|x]$$

- si *f* a suffisamment de capacité pour capter cette relation entre *x* et *y*, et qu'on a suffisamment d'exemples pour généraliser hors de l'ensemble d'apprentissage, bien sûr.

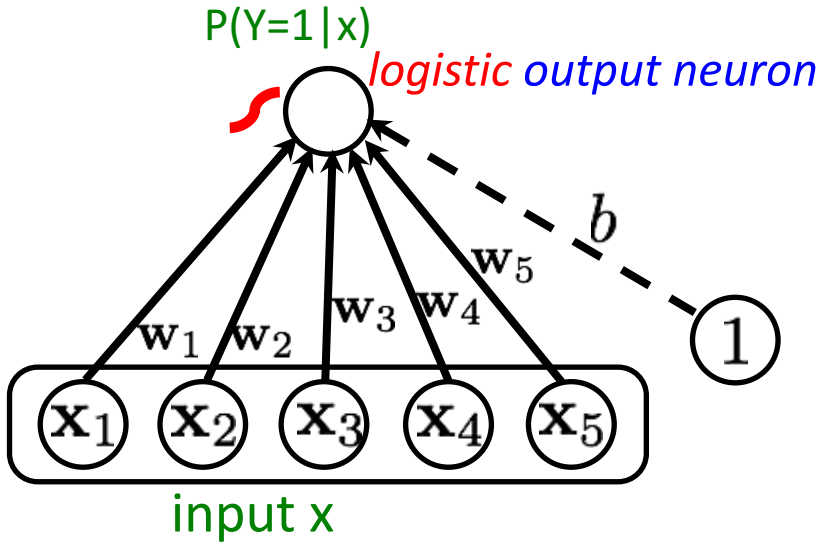
Régression Logistique



- Predict the probability of a **category** *y*, given input *x*
  - *P(Y=y | X=x)*
- Simple extension of linear regression (binary case):
  - *P(Y=1 | X=x)* = sigmoid*(b + w. x)*
- Train by tuning *(b,w)* to maximize average log-likelihood

  Average(*log P(Y=y|X=x)*)

  over training pairs *(x,y)*, by gradient-based optimization
- This is a very **shallow neural network** (no hidden layer)



Neurones cachés

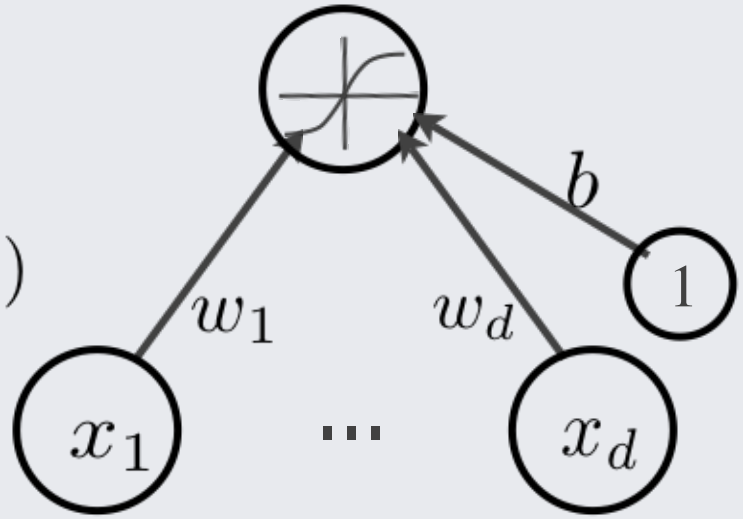
(du cours de Hugo Larochelle)

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

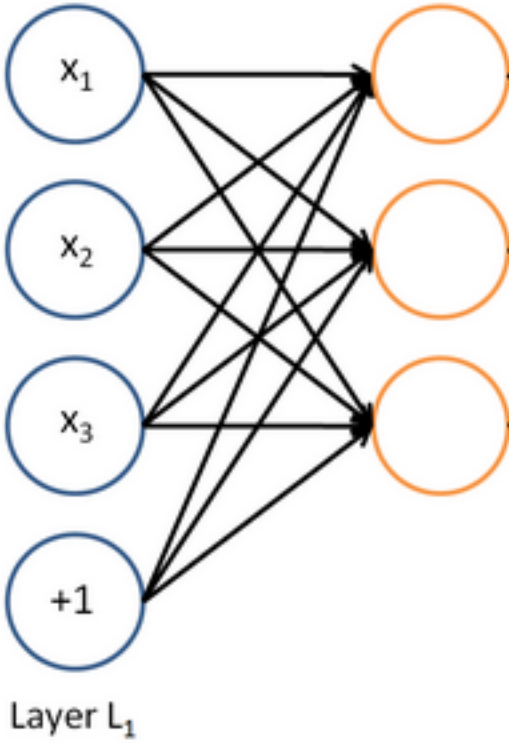
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function

A neural network = running several logistic regressions at the same time

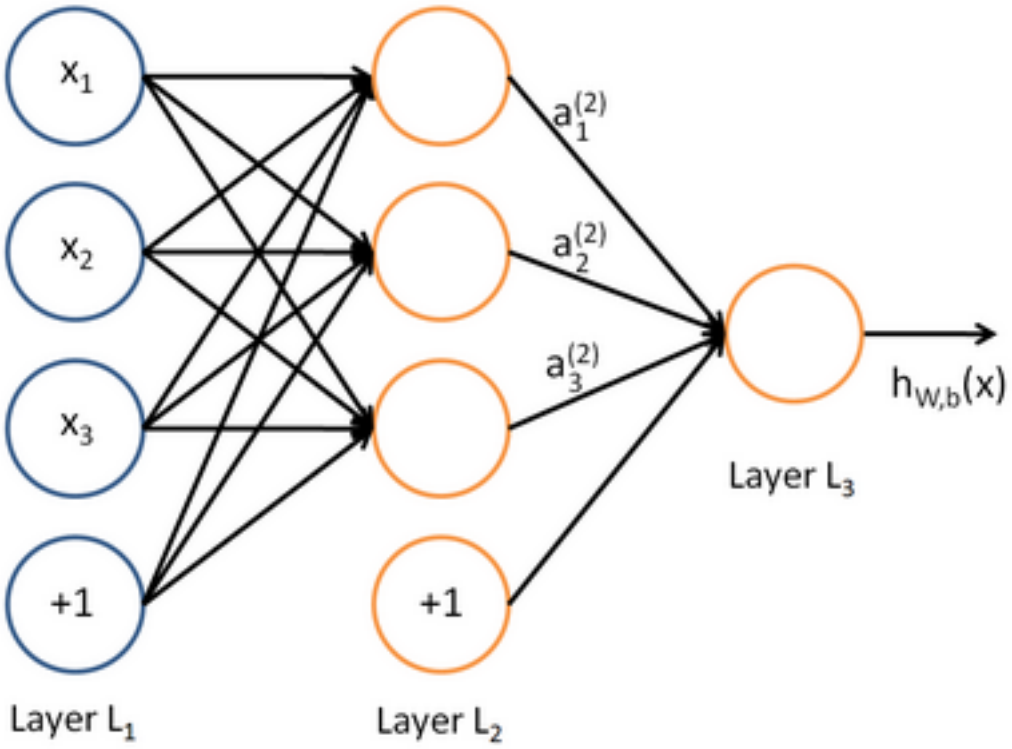
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

A neural network = running several logistic regressions at the same time

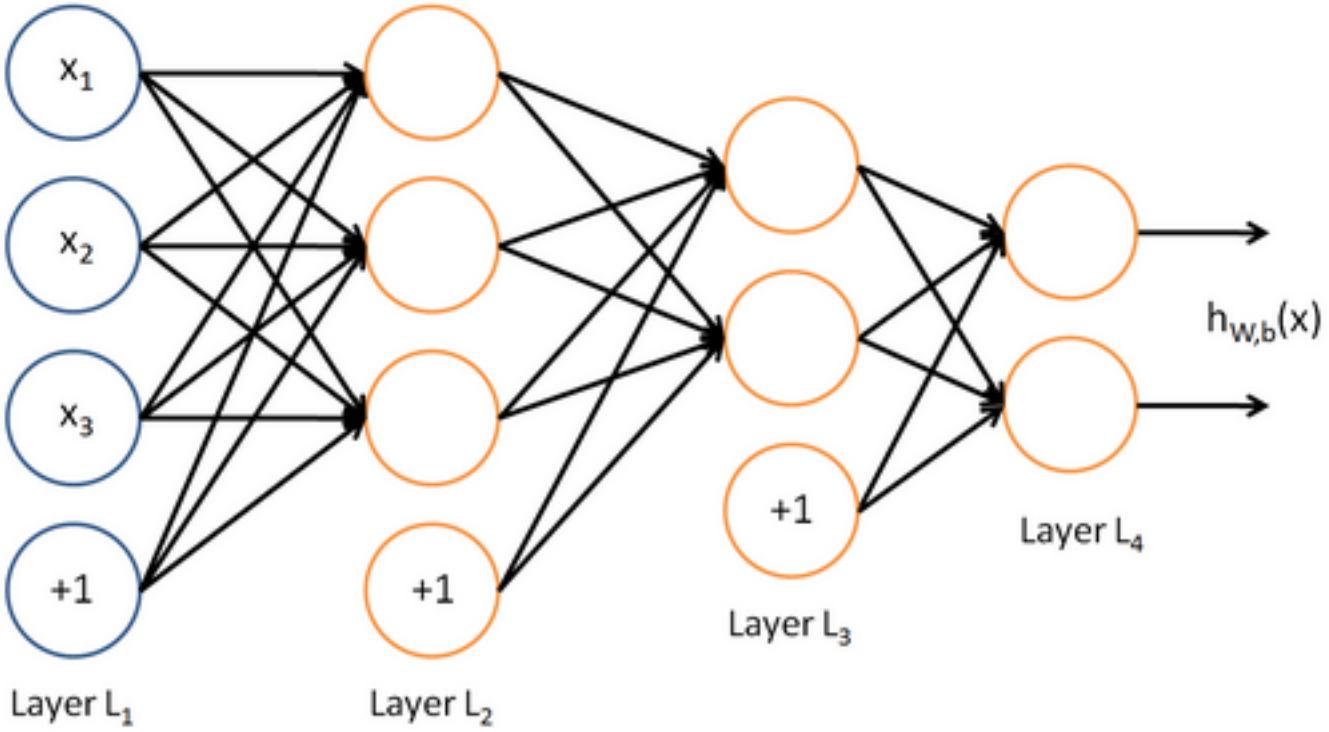
… which we can feed into another logistic regression function



and it is the training criterion that will decide what those intermediate binary target variables should be, so as to make a good job of predicting the targets for the next layer, etc.

A neural network = running several logistic regressions at the same time

- Before we know it, we have a multilayer neural network….

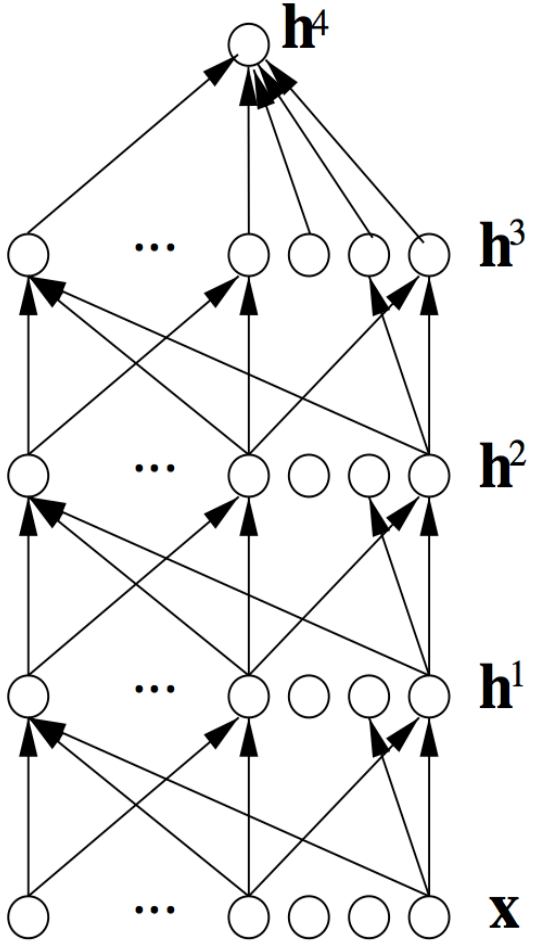


Réseaux de neurones multicouches: approximateur universel

Une série de transformations successives basées sur le même patron mais avec des paramètres différents à chaque couche

Avec seulement 1 couche cachée et **assez d'unités dans la couche**, on a déjà un **approximateur universel**

Plus de couches permet de représenter des fonctions plus complexes avec moins de paramètres



Mais ça ne garantit pas

1. que l'optimisation (l'entraînement) soit facile
2. que ça généralise bien

Non-linéarité = fonction d'activation

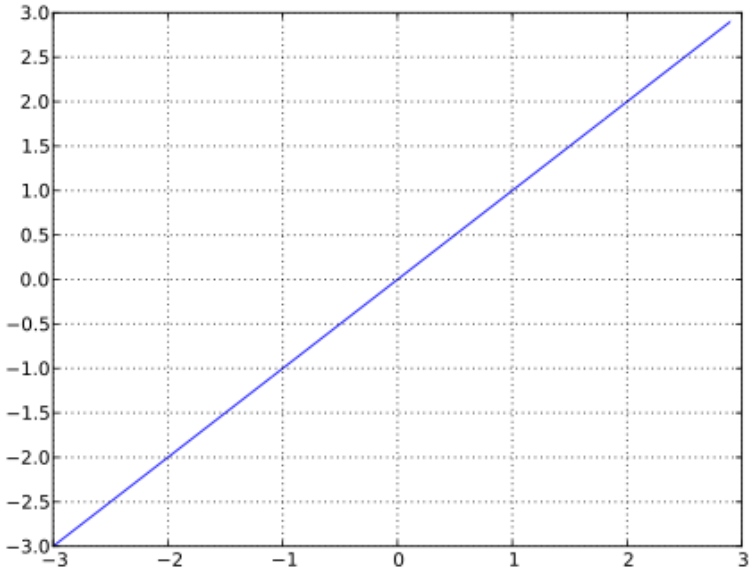
- Si on empile des couches purement linéaires, le résultat reste linéaire en fonction de l'entrée
- Approximateur universel: combiner transformation affine et non-linéarité à chaque couche
- Plusieurs types de non-linéarités sont possibles et préservent la propriété d'approximateur universel
- Plus communes: linéaire (en sortie), sigmoide, tanh, rectifieur (ReLU), softmax
- Entraînement plus facile des réseaux profonds avec les rectifieurs *(Glorot & Bengio AISTATS 2011)*

Fonction d'activation linéaire

(du cours de Hugo Larochelle)

Topics: linear activation function

- Performs no input squashing
- Not very interesting...



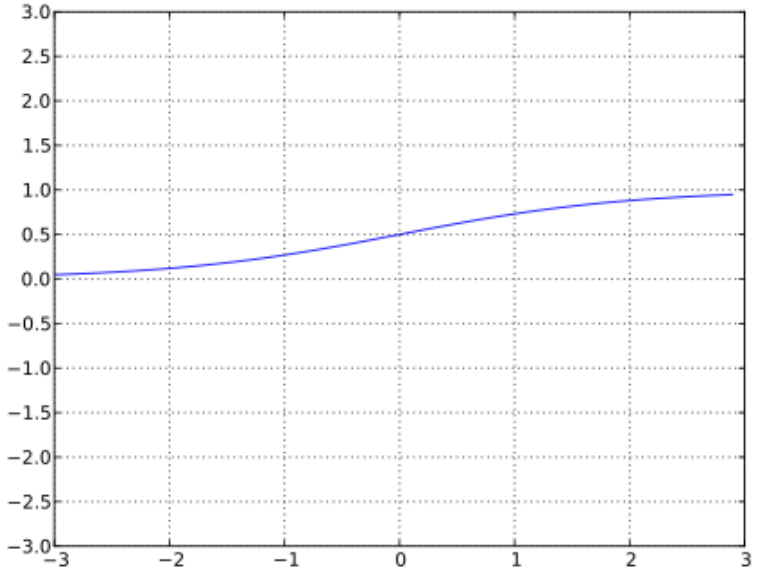
$$g(a) = a$$

Fonction d'activation sigmoide

(du cours de Hugo Larochelle)

Topics: sigmoid activation function

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing



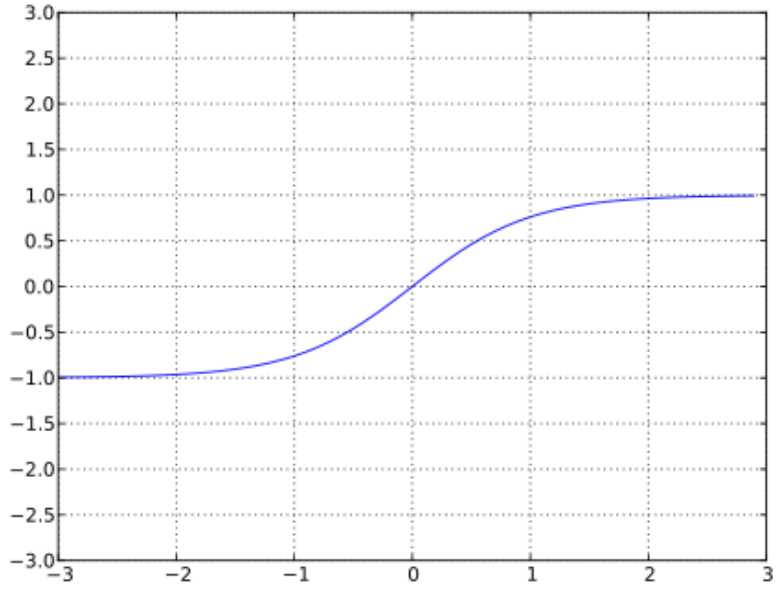
$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

Fonction d'activation tanh

(du cours de Hugo Larochelle)

Topics: hyperbolic tangent ("tanh") activation function

- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing



La tanh est une transformation de la sigmoide:

$$\tanh(x) = 2\,\mathrm{sigmoid}(2\,x) - 1$$

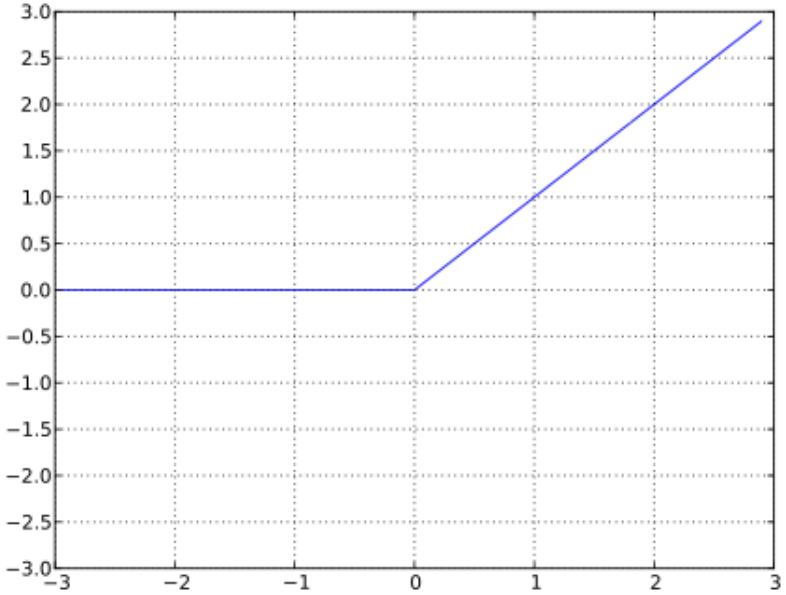
$$g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$$

Fonction d'activation rectificatrice (ReLU)

(du cours de Hugo Larochelle)

Topics: rectified linear activation function

- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities



$$g(a) = \text{reclin}(a) = \max(0, a)$$

Fonction d'activation Softmax

(du cours de Hugo Larochelle)

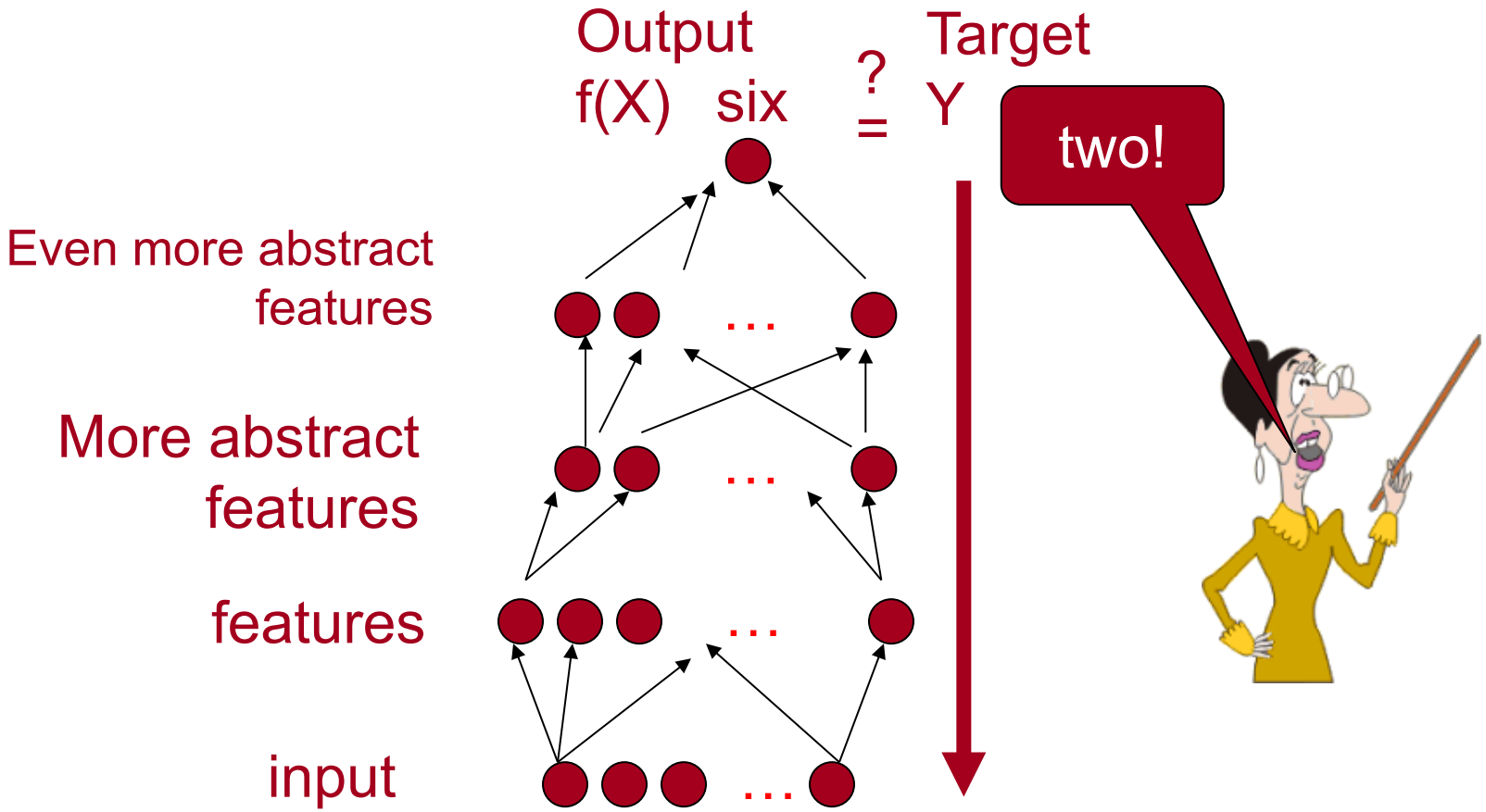
Topics: softmax activation function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$
- We use the softmax activation function at the output:

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

  - strictly positive
  - sums to one
- Predicted class is the one with highest estimated probability

Apprentissage supervisé d'un MLP



Nécessite des paires (X,Y) comme données d'entraînement

Entraînement itératif par SGD

(du cours de Hugo Larochelle)

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\boldsymbol{\theta}$ $(\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\})$
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$$

} training epoch = iteration over **all** examples

- To apply this algorithm to neural network training, we need
  - the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
  - initialization method

Log-vraisemblance comme fonction de perte

(du cours de Hugo Larochelle)

Topics: loss function for classification

- Neural network estimates $f(\mathbf{x})_c = p(y = c|\mathbf{x})$
  - we could maximize the probabilities of $y^{(t)}$ given $\mathbf{x}^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood

natural log (ln)

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

  - we take the log to simplify for numerical stability and math simplicity
  - sometimes referred to as cross-entropy

Log-Likelihood for Neural Nets

- Estimating a conditional probability $P(Y|X)$
- Parametrize it by $P(Y|X) = P(Y|\omega = f_\theta(X))$
- Loss = $-\log P(Y|X)$
- E.g. Gaussian *Y* with parameters $\omega = (\mu, \sigma)$

  typically only μ is the network output, depends on *X*

  Equivalent to MSE criterion (but can generalize it):

  Loss = $-\log P(Y|X) = \log \sigma + ||f_\theta(X) - Y||^2/\sigma^2$
- E.g. Multinoulli *Y* for classification,

$$\omega_i = P(Y = i|x) = f_{\theta,i}(X) = \text{softmax}_i(a(X))$$

Loss = $-\log \omega_Y = -\log f_{\theta,Y}(X)$

Multiple Output Variables

- If they are conditionally independent (given X), then individual prediction LOSSES SIMPLY ADD UP:

$$-\log P(Y|X) = -\log P(Y_1, \ldots Y_k|X) = -\log \prod_i P(Y_i|X) = -\sum_i \log P(Y_i|X)$$

- Likelihood if some Y_i's are missing: just ignore those losses

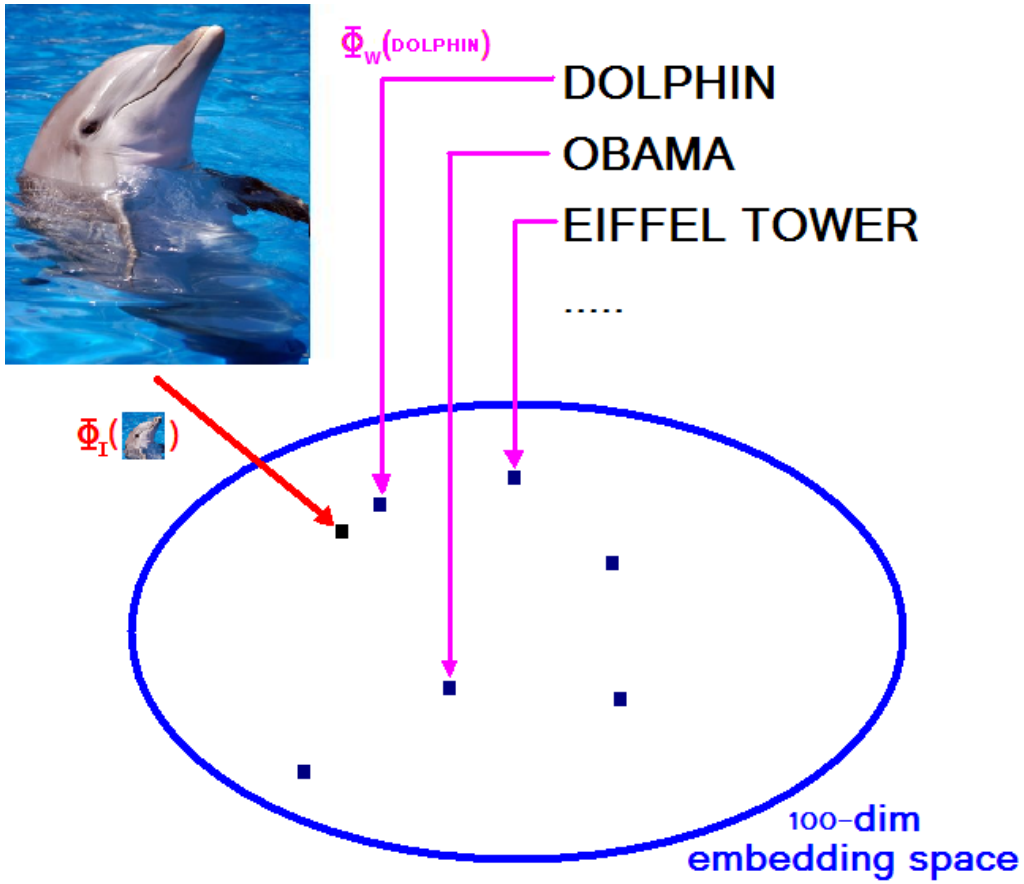
- If not conditionally independent, need to capture the conditional joint distribution $P(Y_1, \ldots Y_k|X)$
  - Example: output = image, sentence, tree, etc.
  - Similar to unsupervised learning problem of capturing joint distribution
  - Exact likelihood may similarly be intractable
  - Example: RNNs, NADE, GANs, VAEs, DAEs, etc.

Google Image Search:
Different object types represented in the same space

Google:

S. Bengio, J. Weston & N. Usunier

(IJCAI 2011, NIPS'2010, JMLR 2010, MLJ 2010)



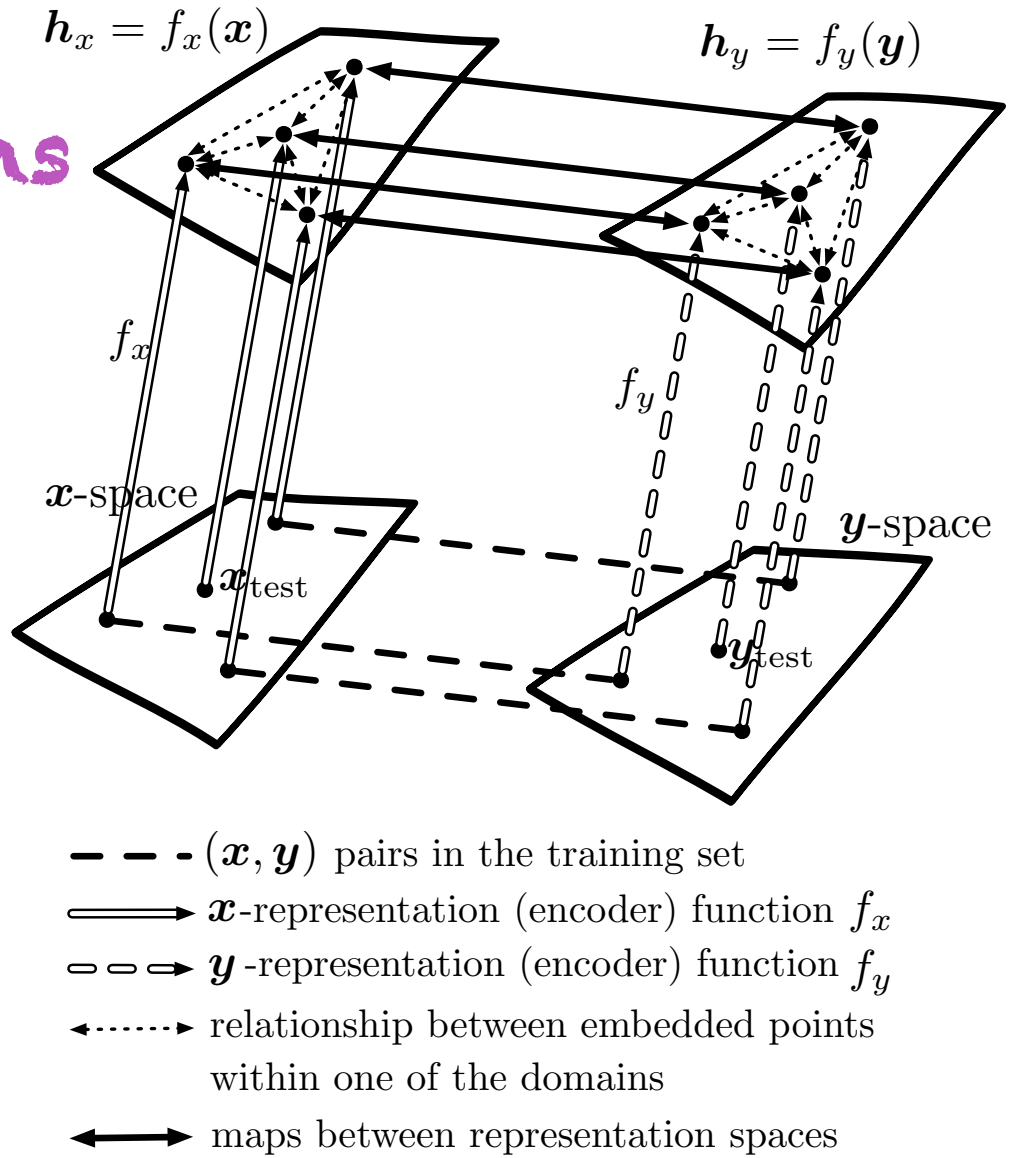
Learn $\Phi_I(\cdot)$ and $\Phi_W(\cdot)$ to optimize precision@k.

Maps Between Representations

x and *y* represent different modalities, e.g., image, text, sound…

Can provide 0-shot generalization to new categories (values of *y*)

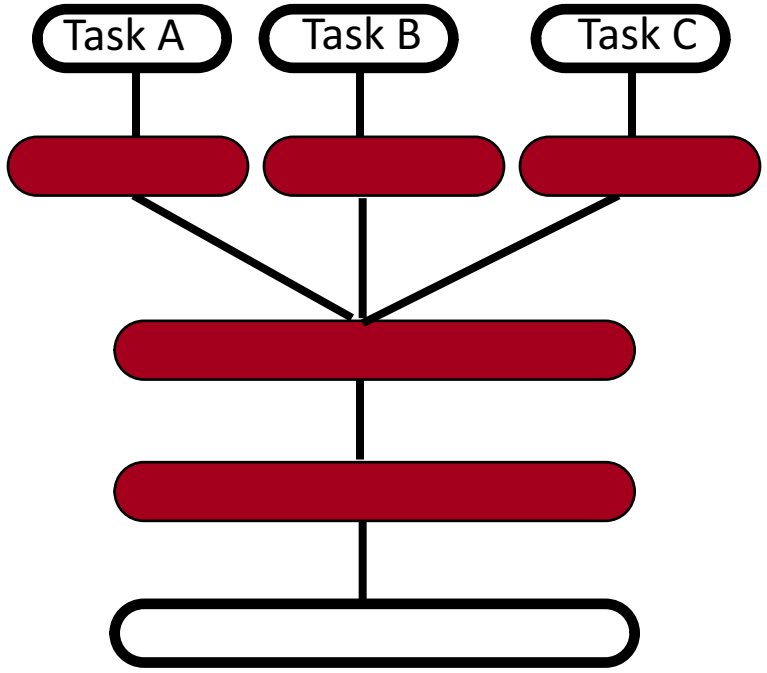
(Larochelle et al AAAI 2008)



Multi-Task Learning

- Generalizing better to new tasks (tens of thousands!) is crucial to approach AI
- Deep architectures learn good intermediate representations that can be shared across tasks

  (Collobert & Weston ICML 2008, Bengio et al AISTATS 2011)
- Good representations that disentangle underlying factors of variation make sense for many tasks because **each task concerns a subset of the factors**



E.g. dictionary, with intermediate concepts re-used across many definitions

Prior: shared underlying explanatory factors between tasks

Combining Multiple Sources of Evidence with Shared Representations

- Traditional ML: data = matrix
- Relational learning: multiple sources, different tuples of variables
- Share representations of same types across data sources
- Shared learned representations help propagate information among data sources: e.g., WordNet, XWN, Wikipedia, **FreeBase**, ImageNet…(Bordes et al AISTATS 2012, ML J. 2013)

- **FACTS = DATA**
- **Deduction = Generalization**