

Deep Feedforward Neural Networks

Yoshua Bengio

August 28-29th, 2017 @ DS3

Data Science Summer School

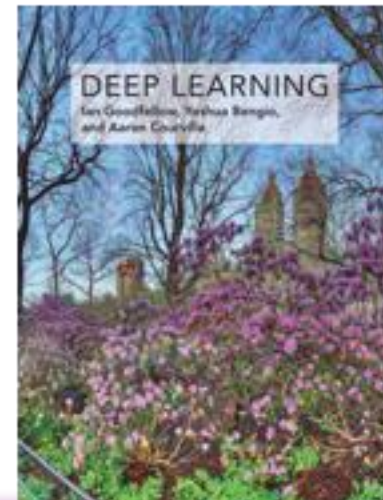


CIFAR
CANADIAN
INSTITUTE
FOR
ADVANCED
RESEARCH

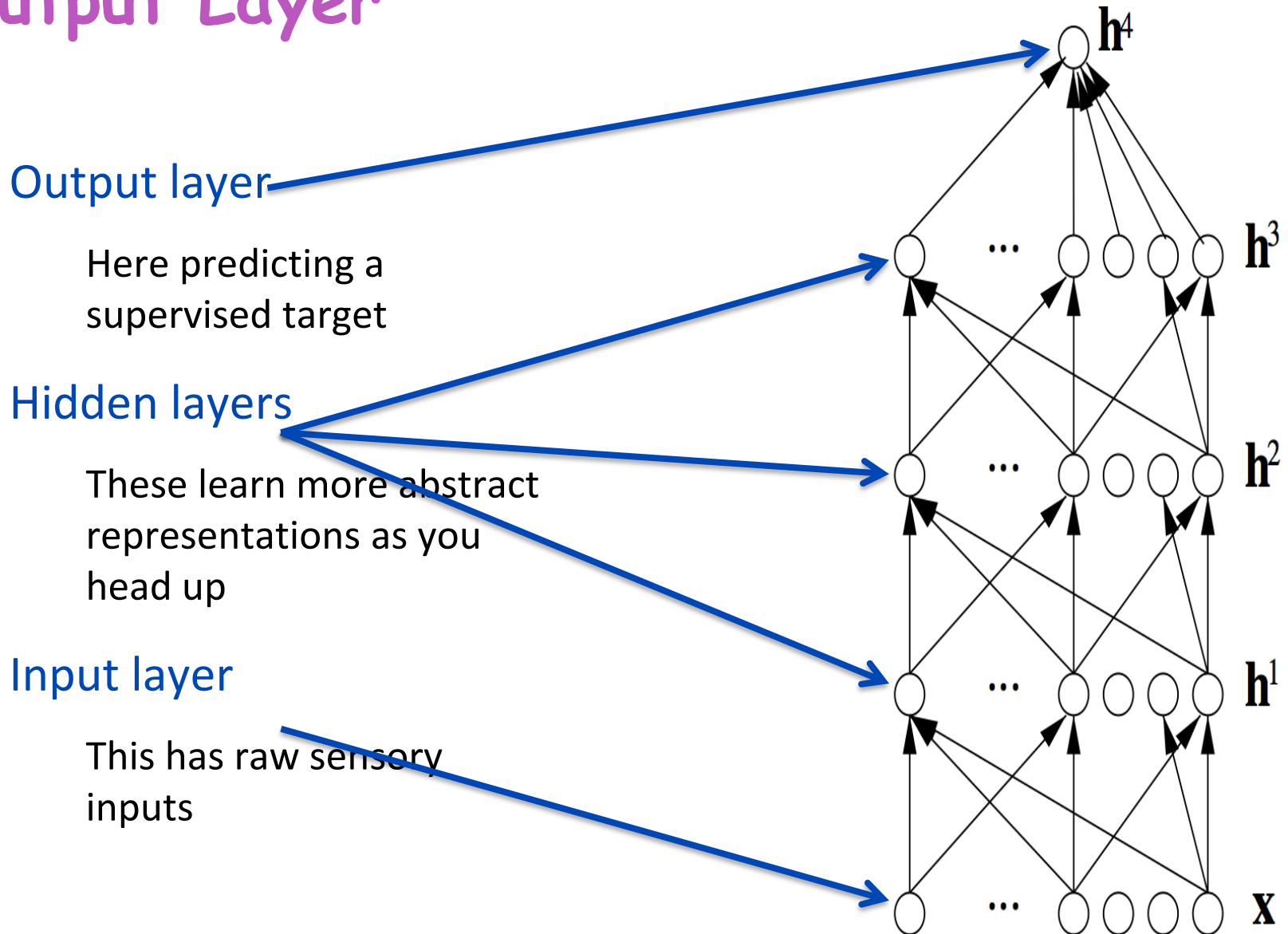
Université 
de Montréal



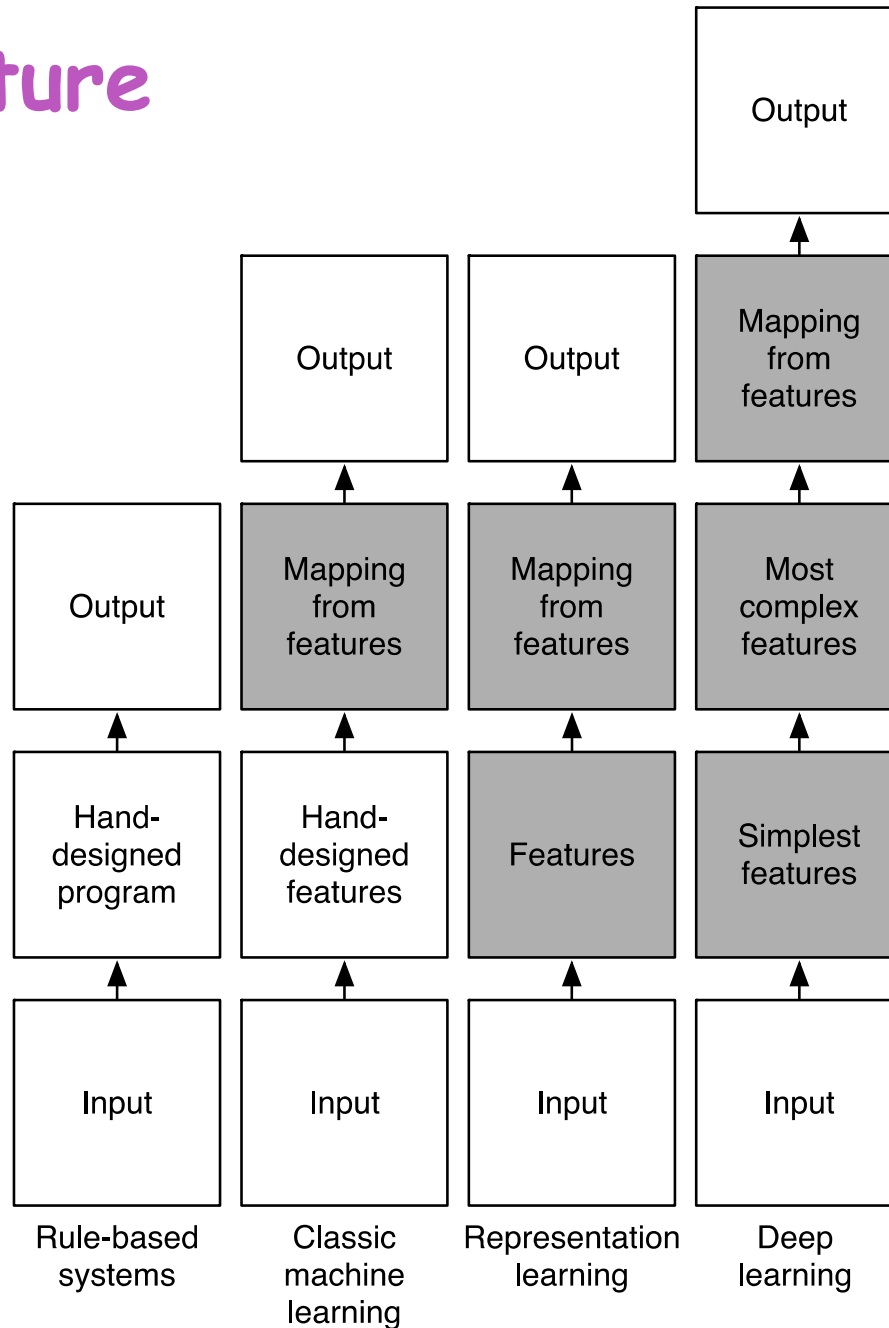
PLUG: **Deep Learning**, MIT Press book is out,
chapters will remain online



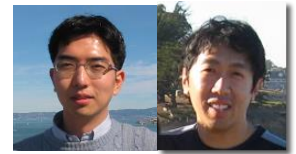
Input Layer, Hidden Layers, Output Layer



Automating Feature Discovery



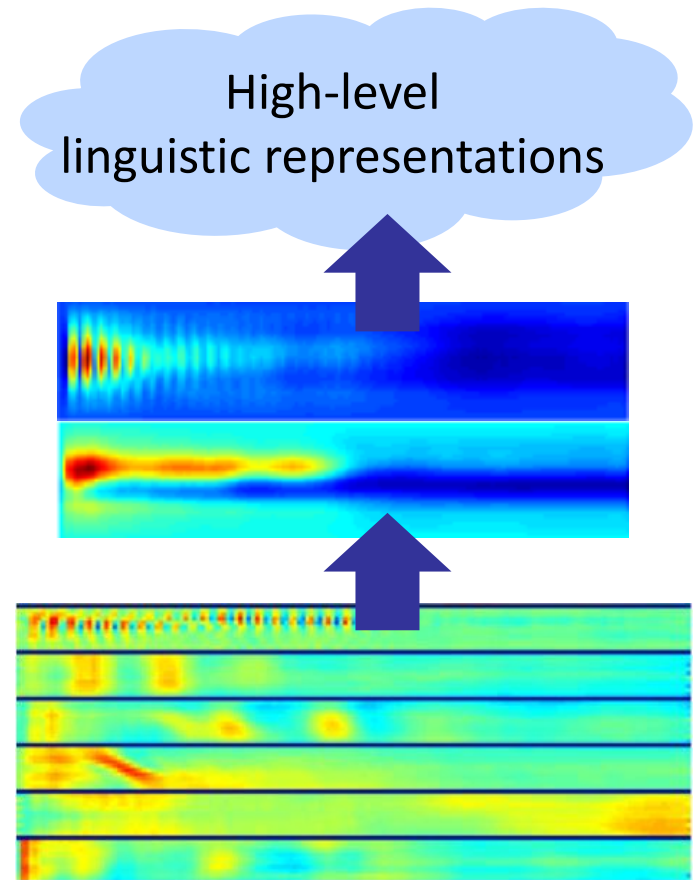
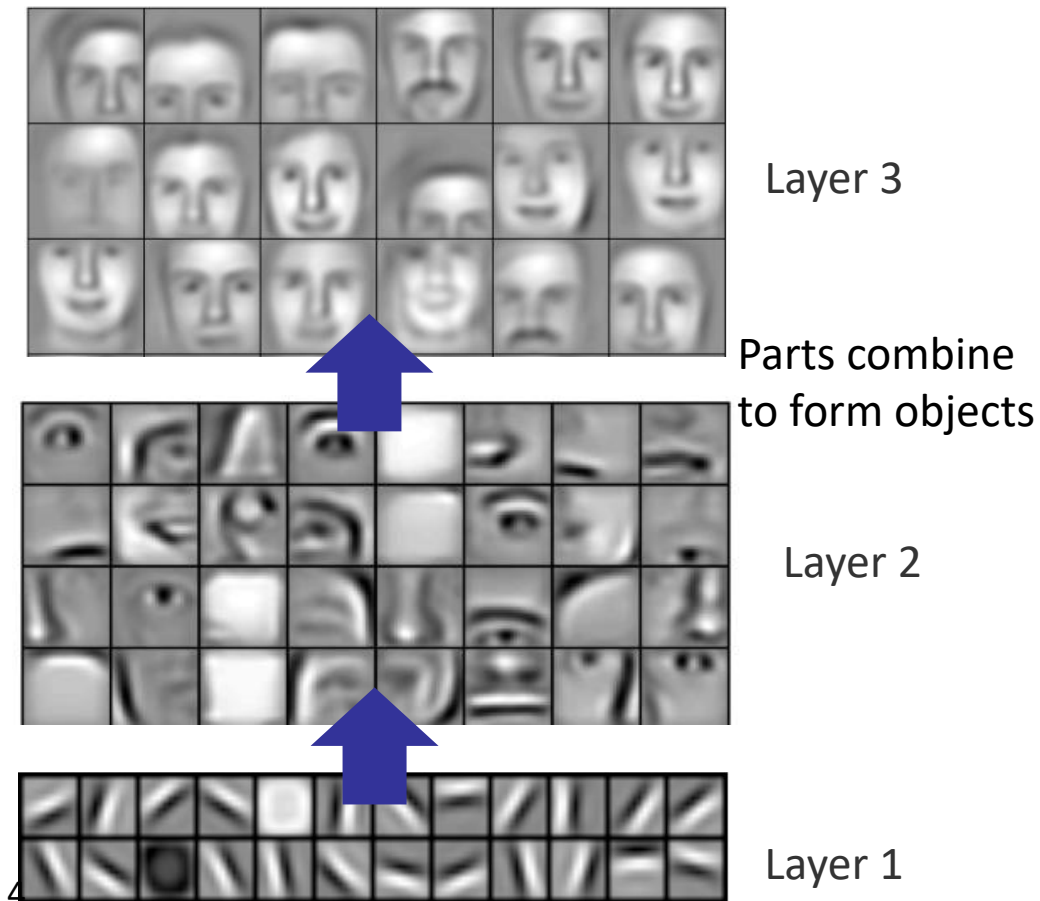
Learning multiple levels of representation



(Lee, Largman, Pham & Ng, NIPS 2009)

(Lee, Grosse, Ranganath & Ng, ICML 2009)

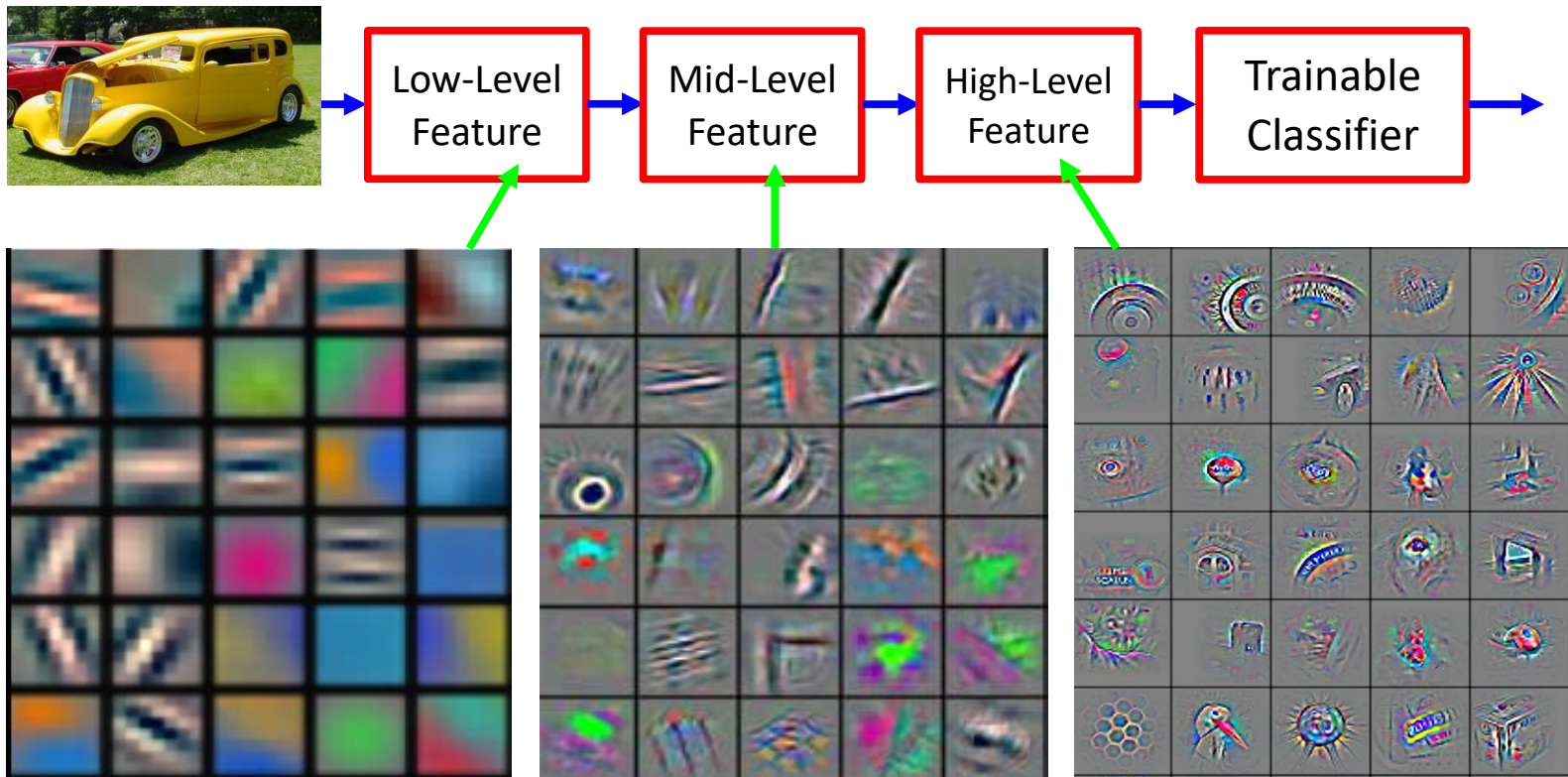
Successive model layers learn deeper intermediate representations



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

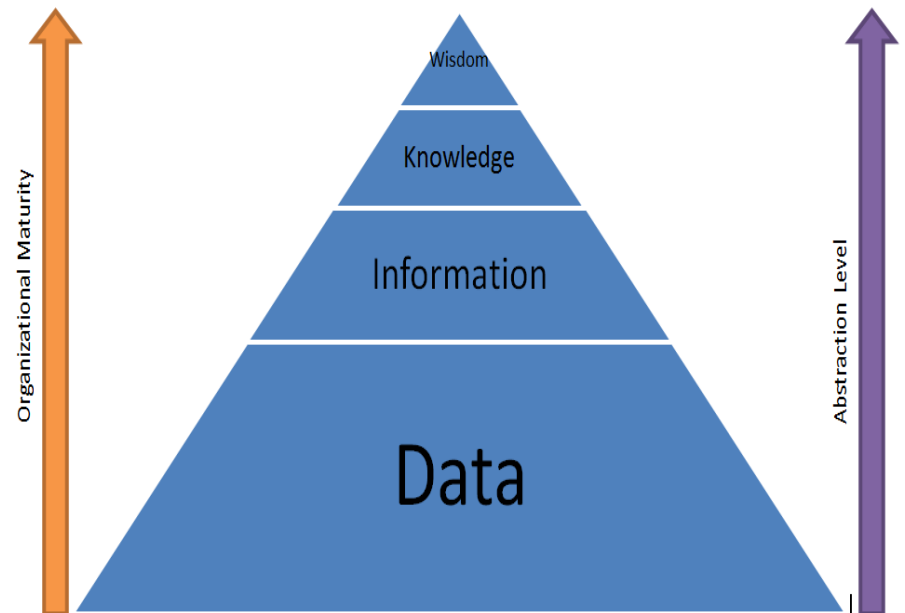
Why Multiple Layers? The World is Compositional

- 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



Learning Multiple Levels of Abstraction

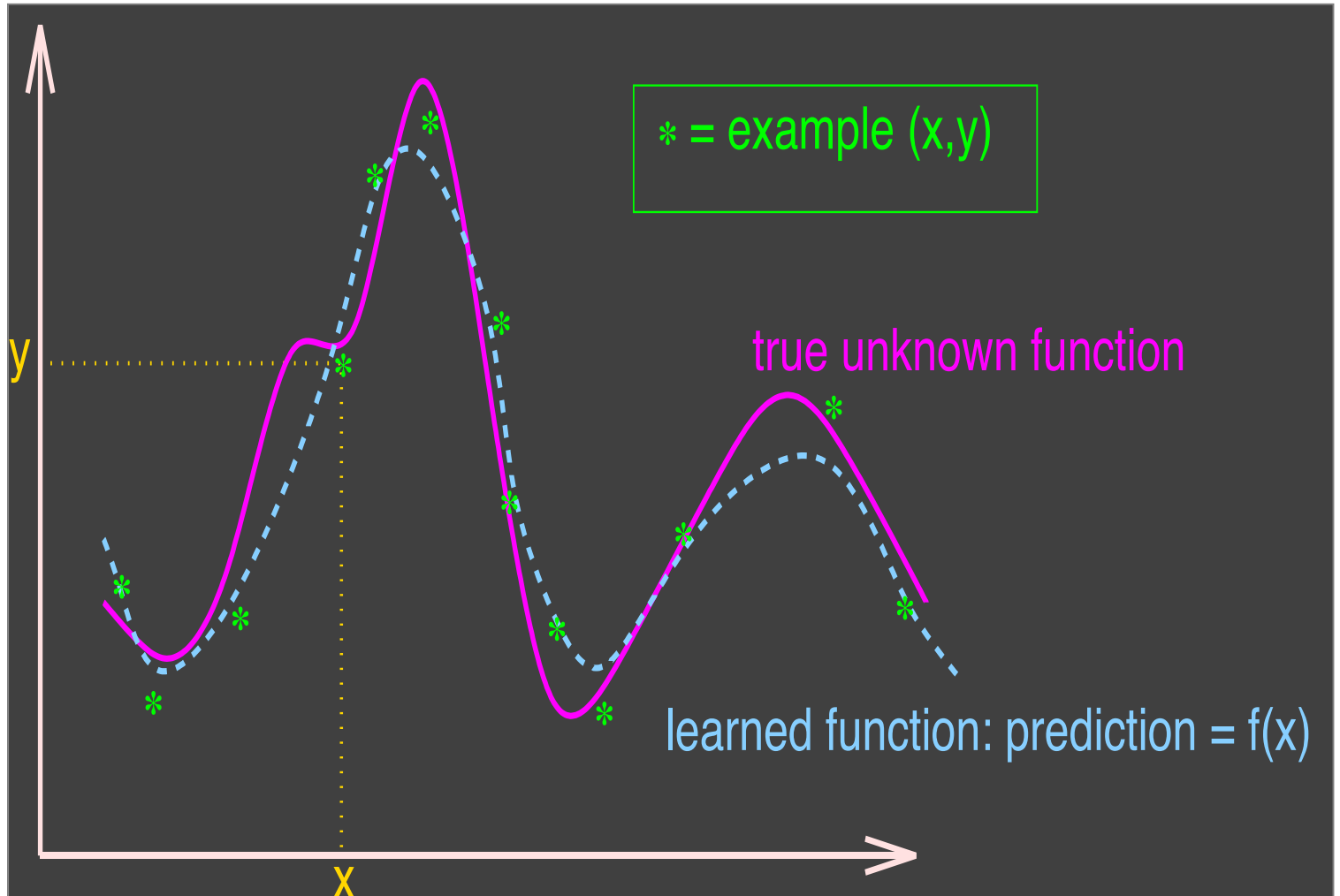
- The big payoff of deep learning is to allow learning higher levels of abstraction
- Higher-level abstractions disentangle the factors of variation, which allows much easier generalization and transfer



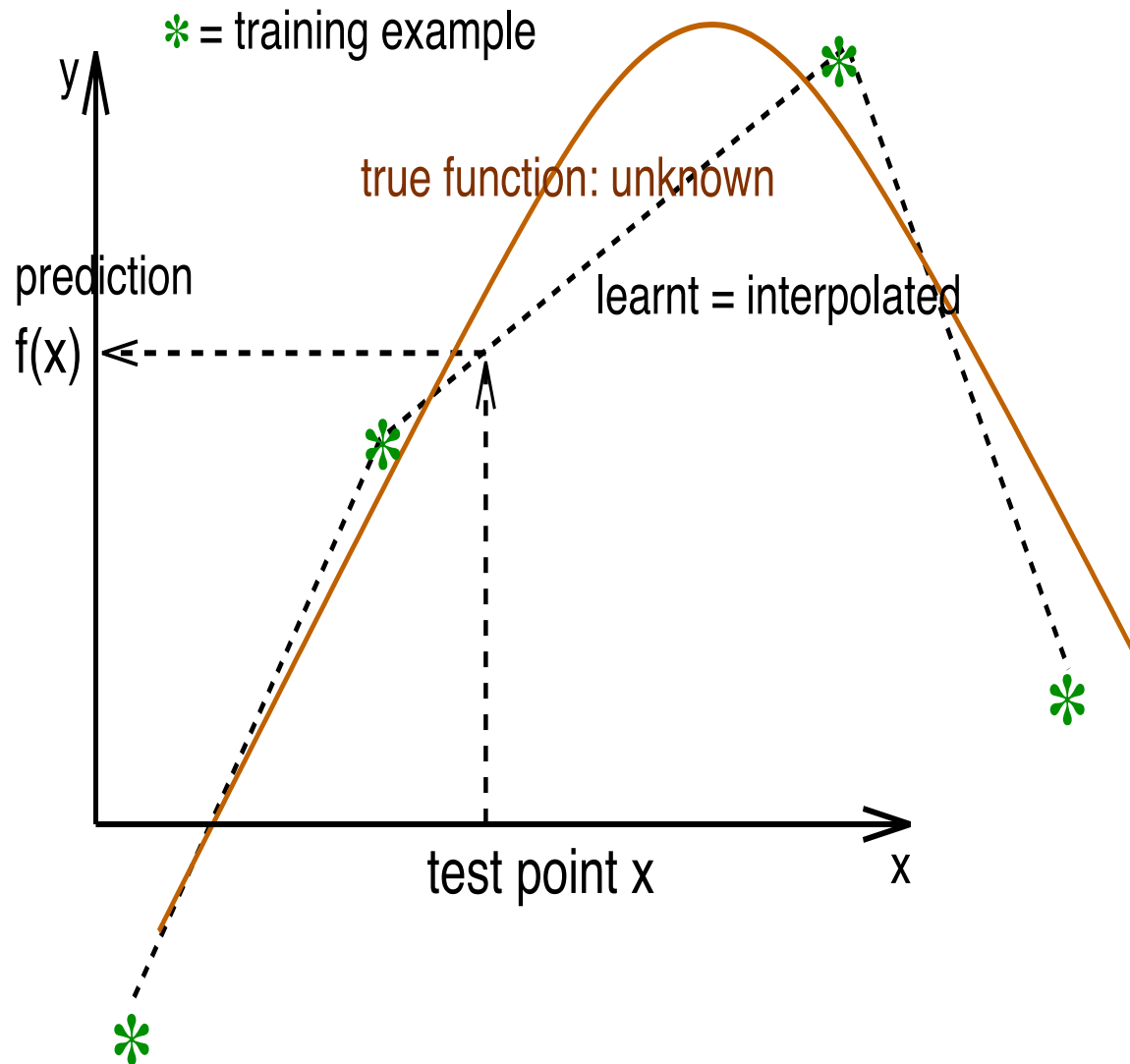
Deep Learning: Learning an Internal Representation

- Unlike other ML methods with either
 - no intermediate representation (linear)
 - or fixed (generally very high-dimensional) intermediate representations (SVMs, kernel machines)
- What is a good representation? Makes other tasks easier.

Easy Learning



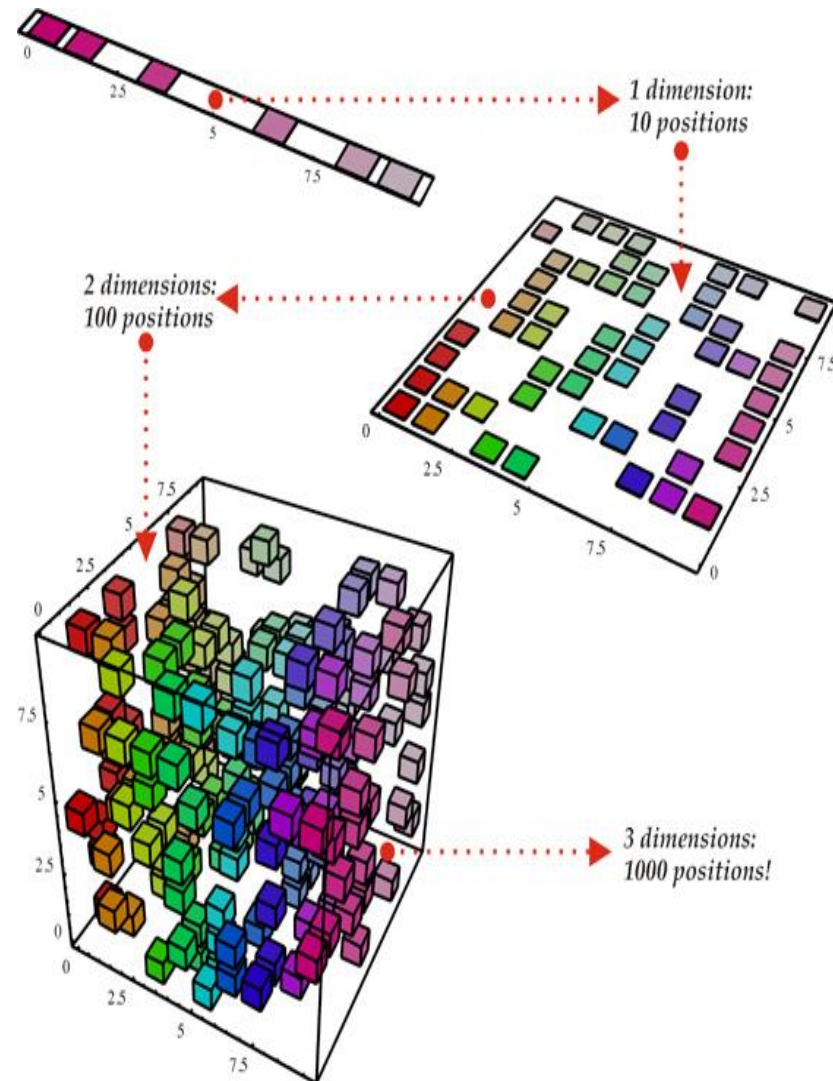
Local Smoothness Prior: Locally Capture the Variations



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

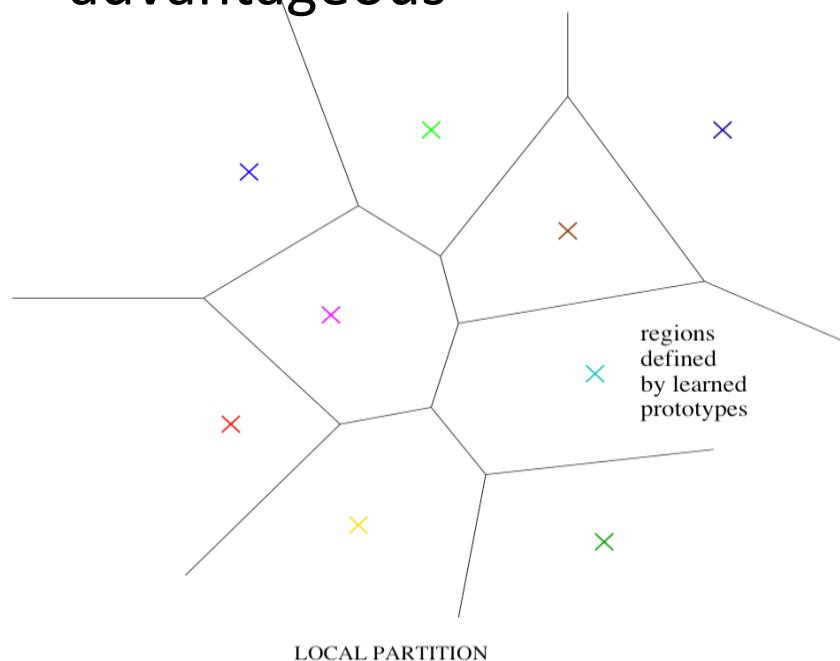
Distributed representations / embeddings: **feature learning**

Deep architecture: **multiple levels of feature learning**

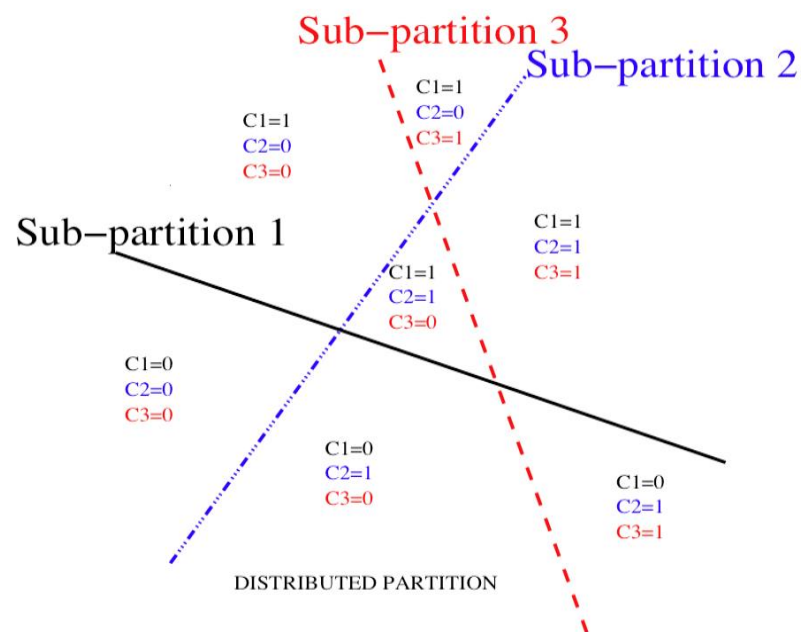
Prior: compositionality is useful to describe the world around us efficiently

Distributed Representations: The Power of Compositionality - Part 1

- Distributed (possibly sparse) representations, learned from data, can capture the **meaning** of the data and state
- Parallel composition of features: can be exponentially advantageous



Not Distributed



Distributed

Deep Representations: The Power of Compositionality - Part 2

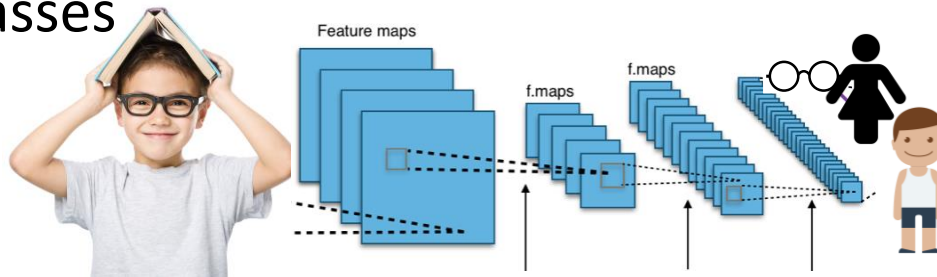
- Learned function seen as a composition of simpler operations, e.g. inspired by neural computation
- Hierarchy of features, concepts, leading to more abstract factors enabling better generalization
- Again, theory shows this can be exponentially advantageous

Why multiple layers? The world is compositional



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.

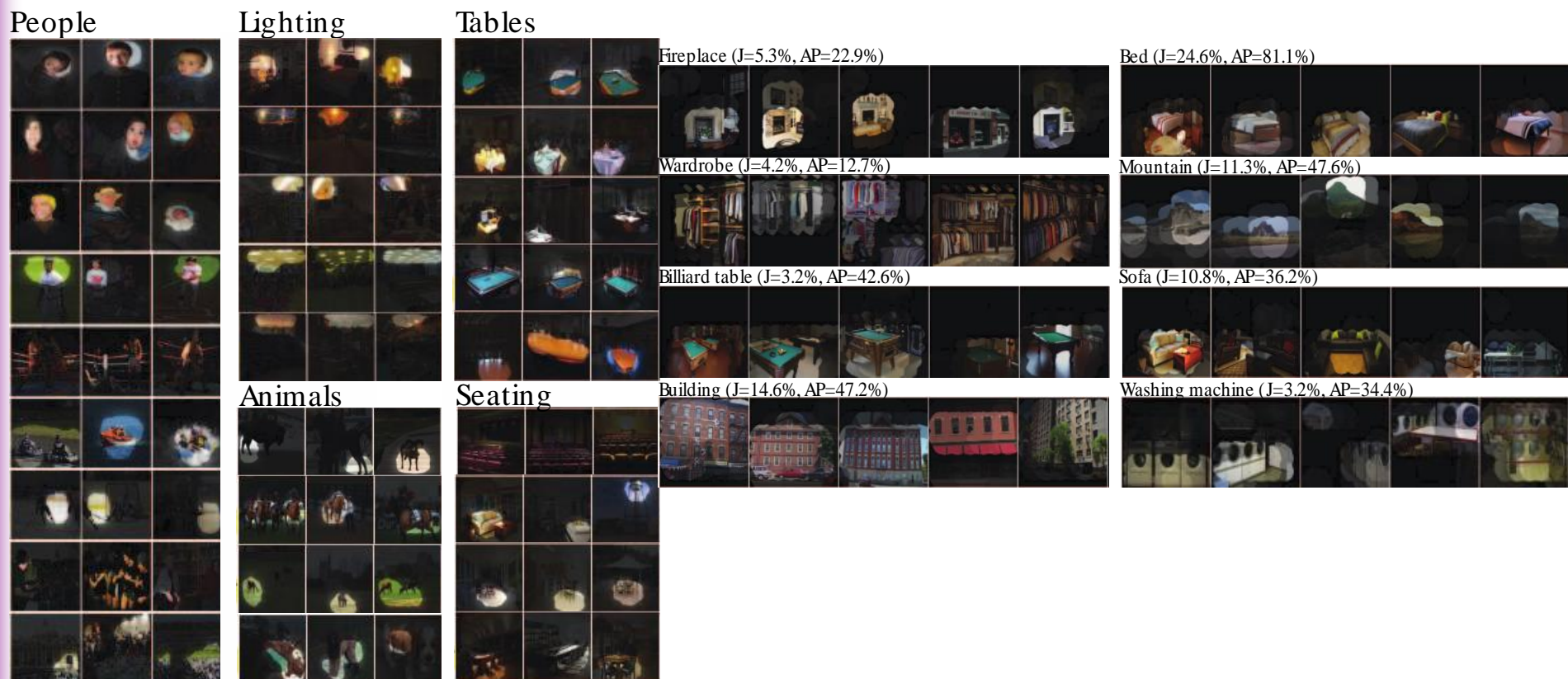


If each of n feature requires $O(k)$ parameters, need $O(nk)$ examples

Non-parametric methods would require $O(n^d)$ examples

Hidden Units Discover Semantically Meaningful Concepts

- *Zhou et al & Torralba, arXiv1412.6856, ICLR 2015*
- *Network trained to recognize places, not objects*

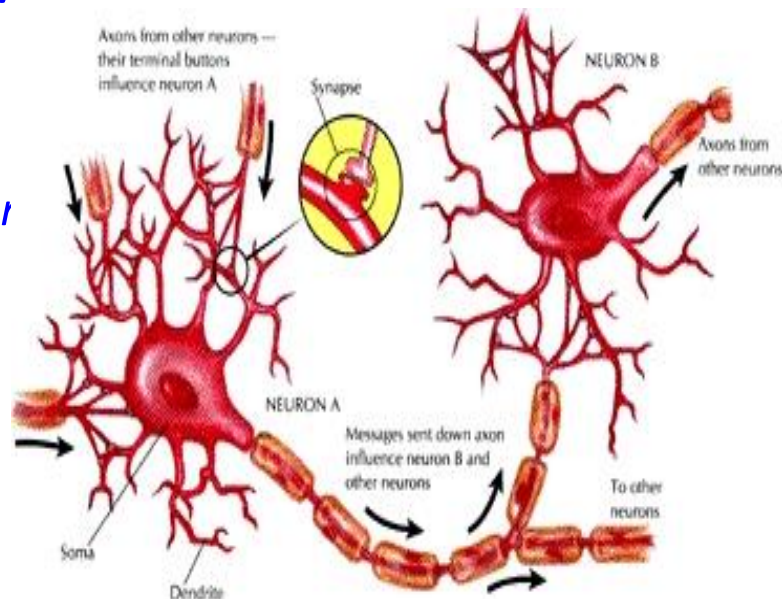
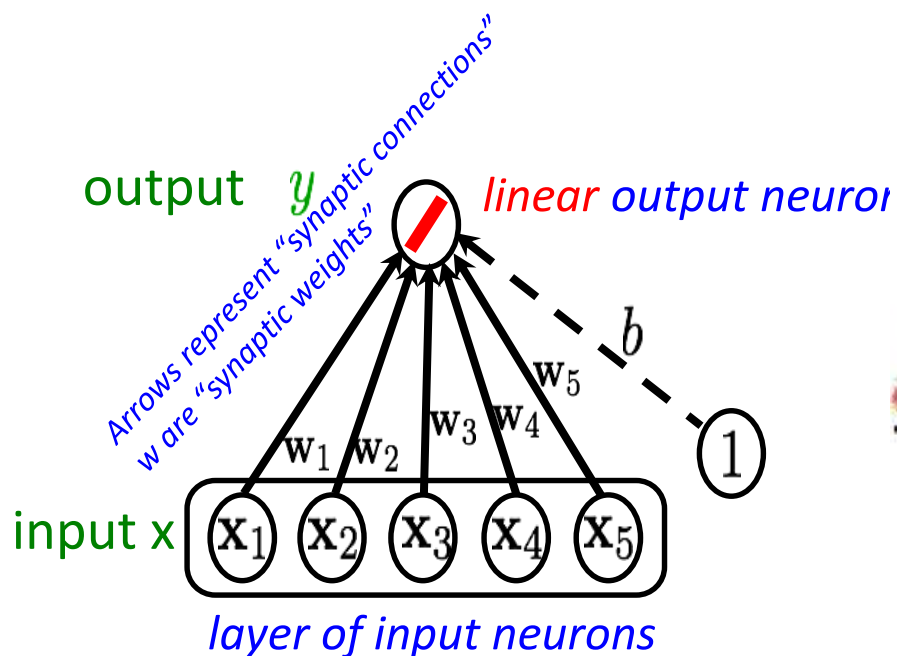


Linear regression, linear neuron

Intuitive understanding of the dot product:
each component of \mathbf{x} weighs differently on the response.

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

Neural network terminology:



Estimating a conditional expectation with the quadratic loss

Typically, to predict a continuous quantity, minimize quadratic loss

- If the loss function is $L = (y - f_{\theta}(x))^2$

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

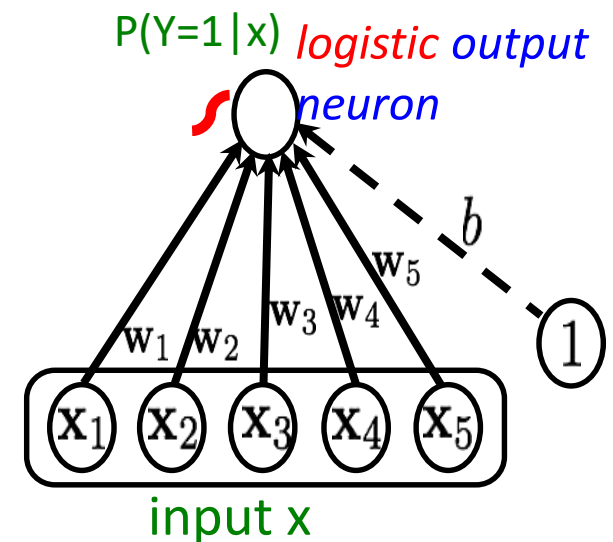
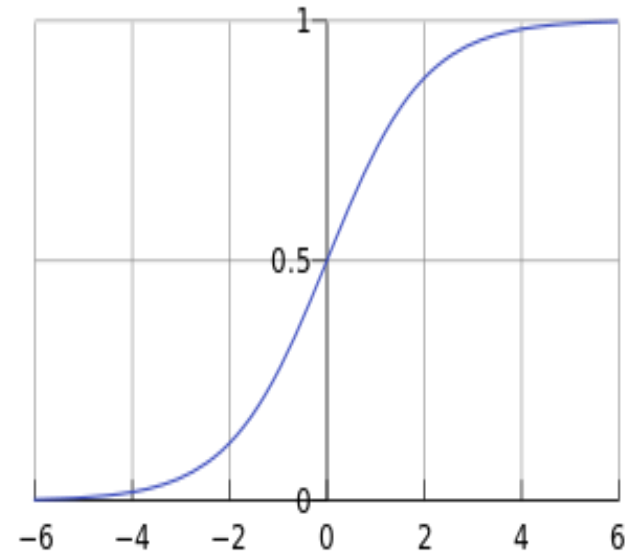
- then the network estimates

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

- if f has enough capacity and data (and training) to capture this

Logistic Regression

- Predict the probability of a **category** y , given input x
 - $P(Y=y \mid X=x)$
- Simple extension of linear regression (binary case):
 - $P(Y=1 \mid X=x) = \text{sigmoid}(b + w \cdot x)$
- Train by tuning (b, w) to maximize average log-likelihood
Average($\log P(Y=y \mid X=x)$)
over training pairs (x, y) , by gradient-based optimization
- This is a very **shallow neural network** (no hidden layer)



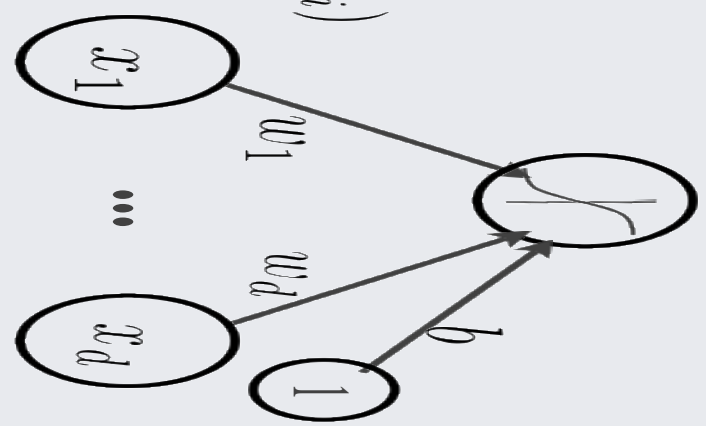
Hidden units

- Neuron pre-activation (or input activation):

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

- Neuron (output) activation

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



- \mathbf{w} are the connection weights

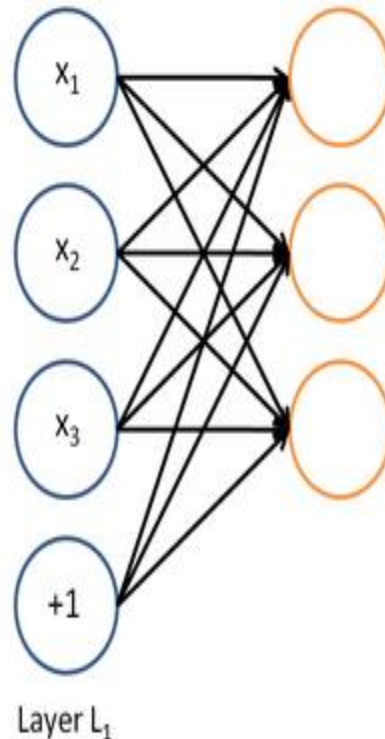
- b is the neuron bias

- $g(\cdot)$ is called the activation function

(from
Hugo
Larochelle)

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

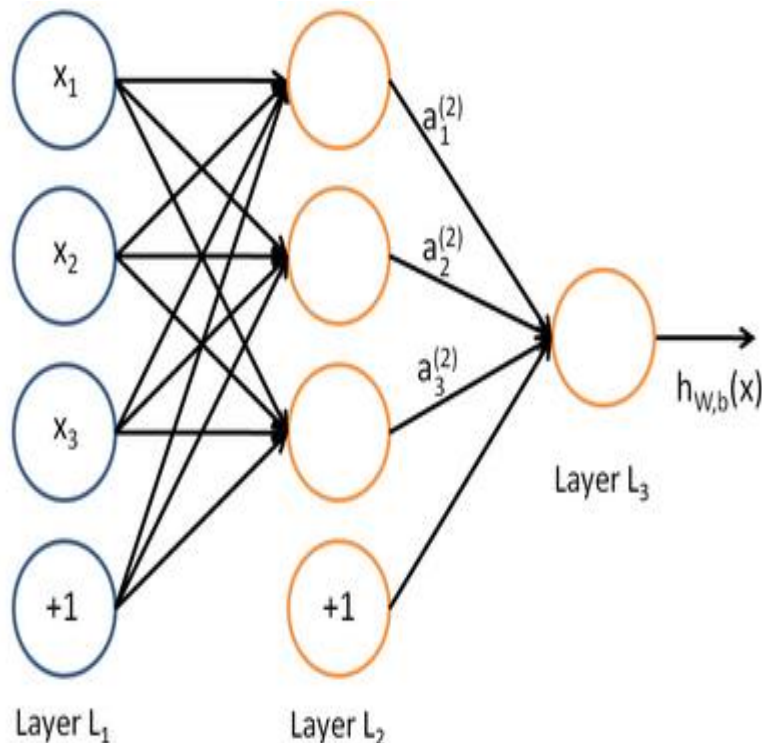
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



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

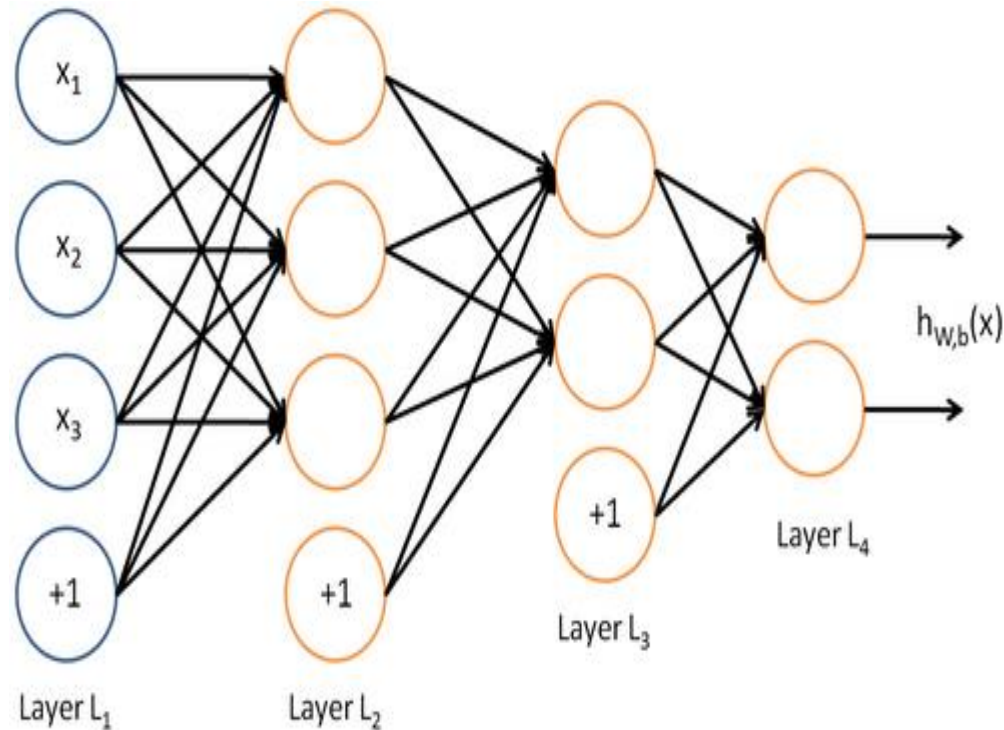
... which we can feed into another logistic regression function



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....



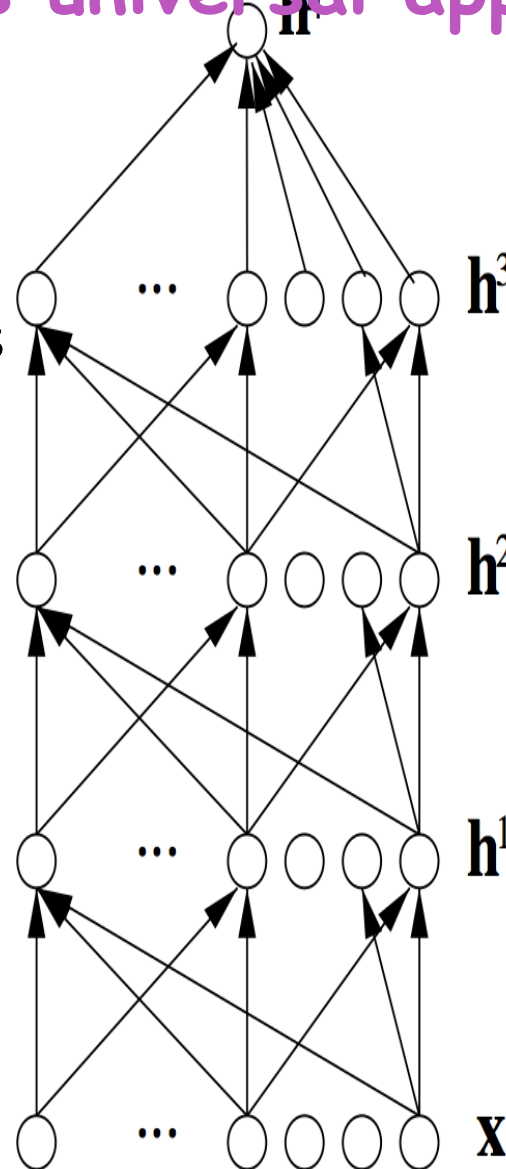
Multilayer network as universal approximator

A series of non-linear transformations of the same type but different parameters

A single but large enough hidden layer yields a

universal approximator

More layers allow representing more complex functions with less parameters



Universal approximator property does not guarantee

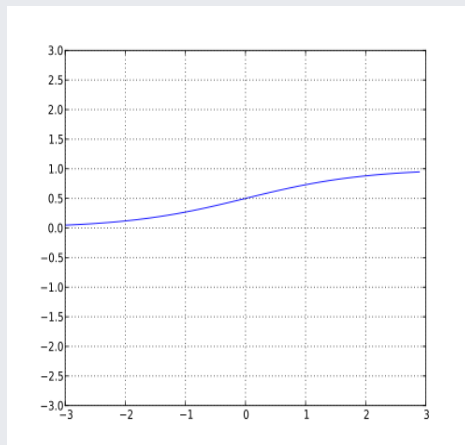
1. easy optimization (low training error is found)
2. good generalization

Non-linearity = activation function

- Stacking linear layers: like one (factorized) linear layer
- Universal approximator : stack linear+nonlinear transformations
- Many types of non-linearities are possible: activation function
 - E.g. linear, sigmoid, tanh, rectifier (ReLU), softmax
- *Breakthrough in 2011: it is much easier to train a deep multilayer network with rectifiers than with sigmoid or tanh, making it possible to train deep nets in a purely supervised way for the first first time (Glorot & Bengio AISTATS 2011)*

Topics: sigmoid activation function

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



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

Topics: softmax activation function

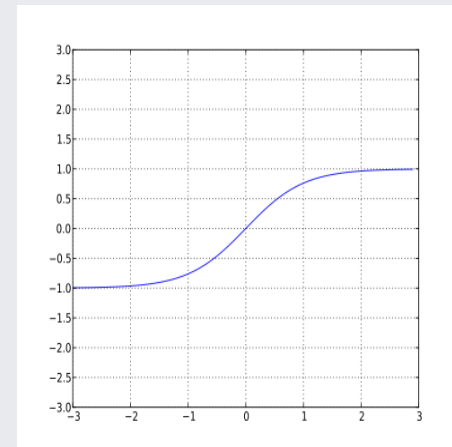
- For multi-class classification:
 - we need multiple outputs (1 output per class)
 - we would like to estimate the conditional probability $p(y = c | \mathbf{x})$
- We use the softmax activation function at the output:

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

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

Topics: hyperbolic tangent ("tanh") activation function

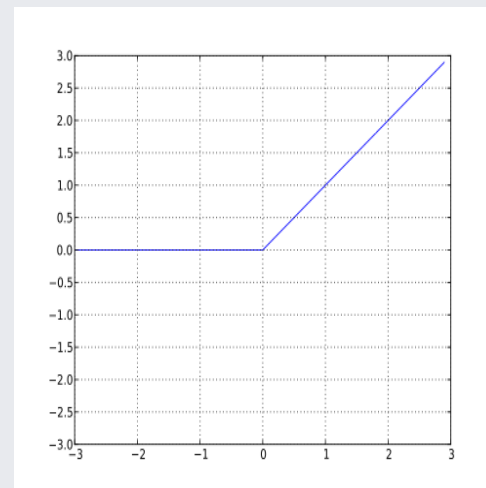
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing



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

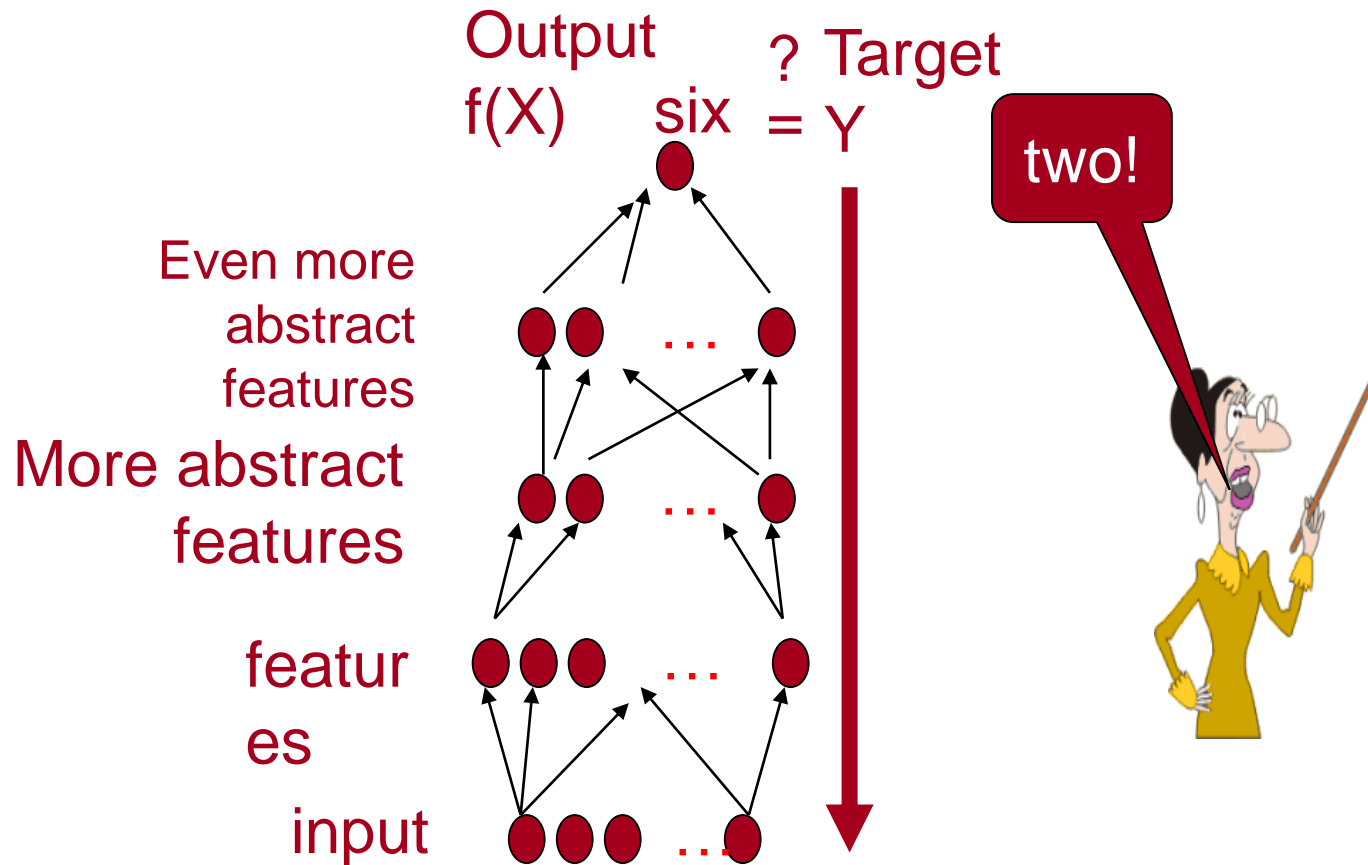
Topics: rectified linear activation function

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



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

Supervised training of an MLP



Requires $(X, Y) = (\text{input}, \text{target})$ pairs as training data

Iterative training by SGD

(from
Hugo
Larochelle)

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- initialize θ ($\theta \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)

- for N iterations

- for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

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

- $$\theta \leftarrow \theta + \alpha \Delta$$

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

- To apply this algorithm to neural network training, we need

- the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \theta), y^{(t)})$

- a procedure to compute the parameter gradients $\nabla_{\theta} l(\mathbf{f}(\mathbf{x}^{(t)}; \theta), y^{(t)})$

- the regularizer $\Omega(\theta)$ (and the gradient $\nabla_{\theta} \Omega(\theta)$)

- initialization method


Log-likelihood as loss function

(from
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 , $\omega = (\mu, \sigma)$

typically only μ is the network output, depends on X

Equivalent to MSE criterion:

$$\text{Loss} = -\log P(Y|X) = \log \sigma + ||f_\theta(X) - Y||^2 / \sigma^2$$

- E.g. Multinoulli Y for classification,

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

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

Multiple Output Variables

- If they are conditionally independent (given X), the individual prediction losses add up:

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

- Likelihood if some Y_i 's are missing: just ignore those losses
- If not conditionally independent, need to capture the conditional joint distribution $P(Y_1, \dots, Y_k|X)$
 - Example: output = image, sentence, tree, etc.
 - Similar to unsupervised learning problem of capturing joint
 - Exact likelihood may similarly be intractable, depending on model

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

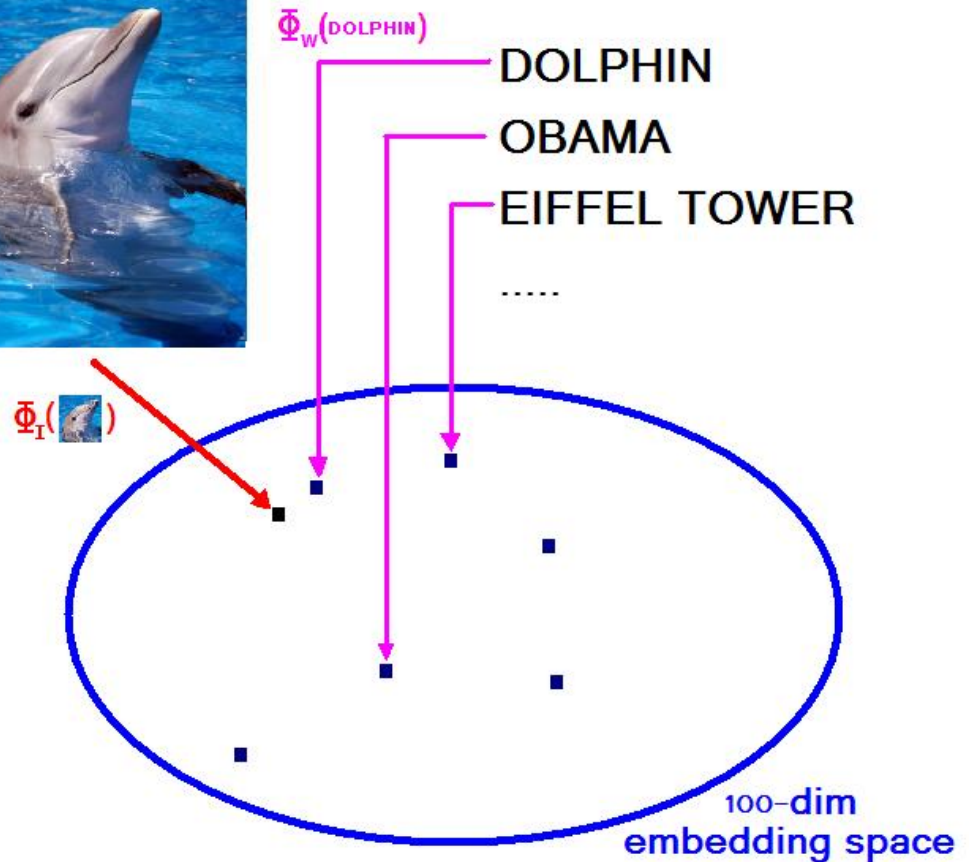


Google:

S. Bengio, J.
Weston & N.
Usunier



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



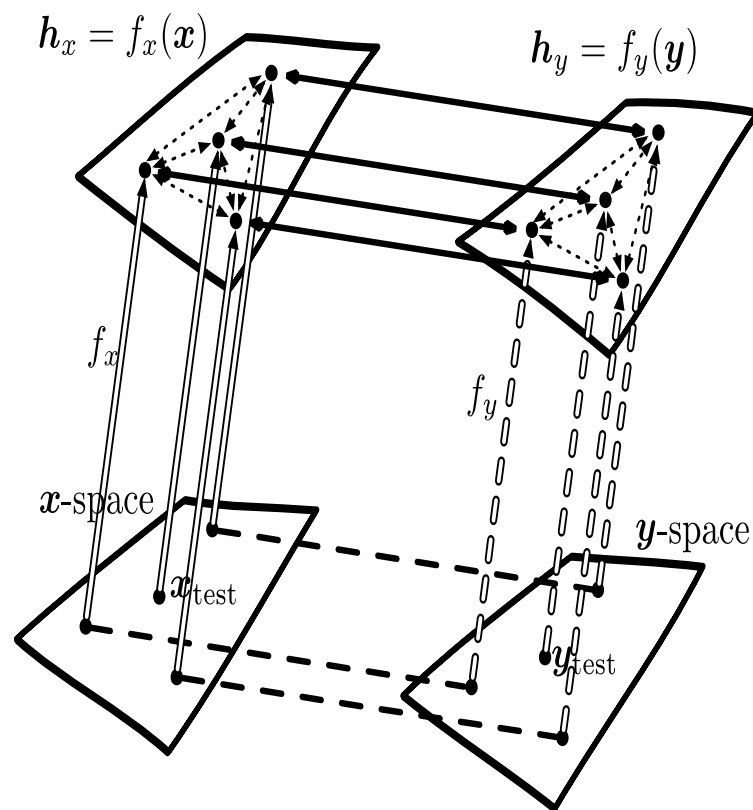
Learn $\Phi_I(\cdot)$ and $\Phi_W(\cdot)$ to optimize precision@k.

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)



--- (x, y) pairs in the training set

\implies x -representation (encoder) function f_x

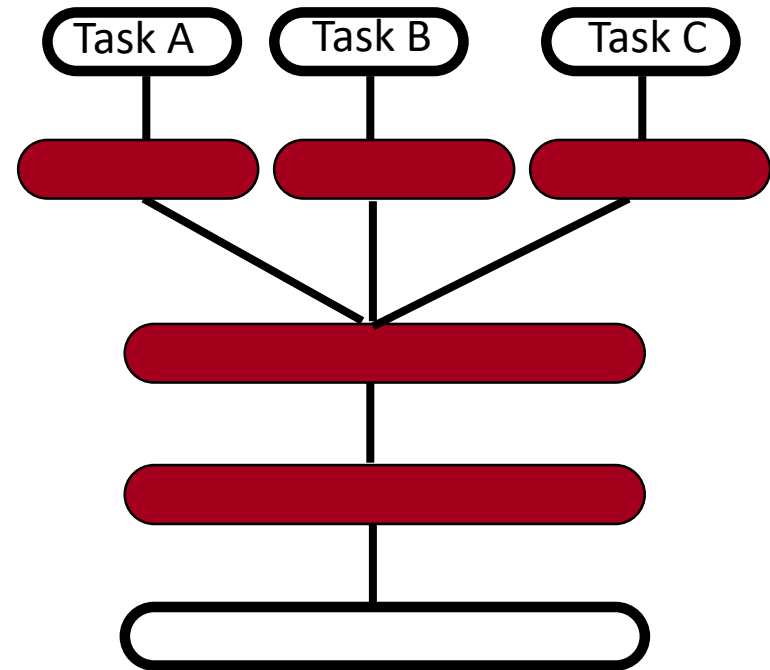
\implies y -representation (encoder) function f_y

\longleftrightarrow relationship between embedded points within one of the domains

\longleftrightarrow maps between representation spaces

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**

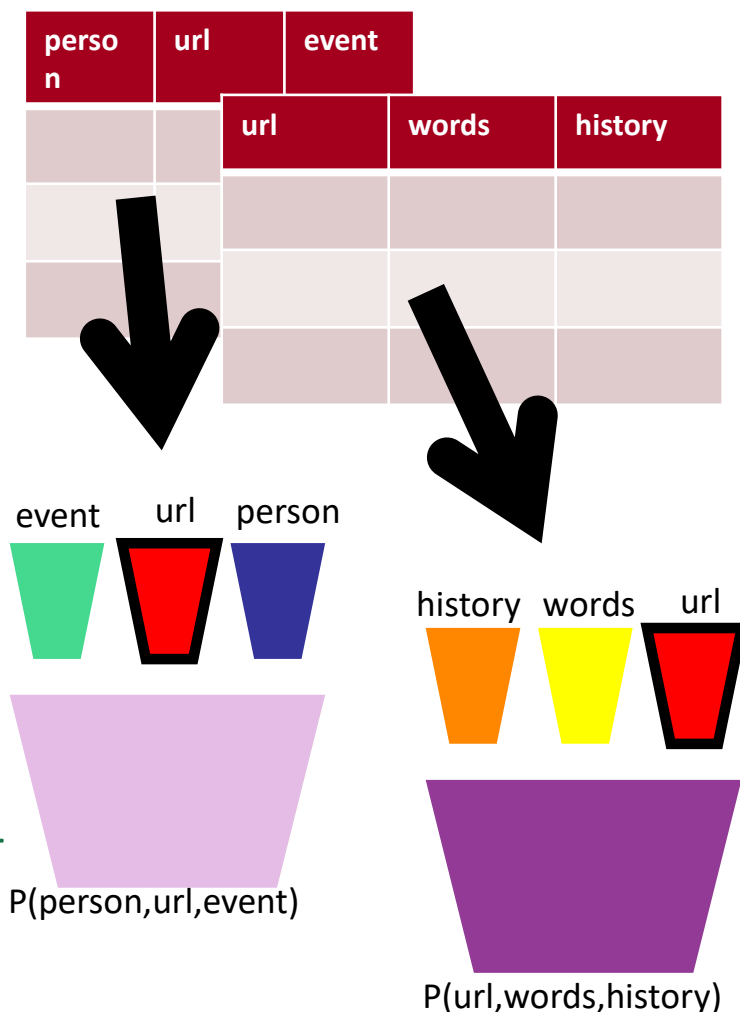


E.g. dictionary, with intermediate concepts re-used across many definitions

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**

34 **Deduction = Generalization**